自　序

我们迎来了新世纪的黎明。21世纪犹如喷薄欲出的一轮红日呈现在人们面前。在这世纪更迭的神圣时刻，人们正满怀激情地发问：新世纪的科学会有什么样的新风采？

新世纪呼唤新科学

回顾已经过去的20世纪，谁也不会否认这样的事实：这个世纪的科学技术取得了惊人的成就，在科学史上谱写了辉煌的篇章。

在20世纪末，具有远见卓识的学者们意识到人类科学正面临着一个新的转折点。1984年，在诺贝尔奖获得者P.W.Anderson、M.Gell-Mann和K.S.Arrow等人的支持下，美国一批从事物理、经济、生物和计算机等学科研究的学者们创建了著名的桑塔费研究所SFI，试图探索未来科学的思维方式。SFI首任所长G.A.Cowan尖锐地指出：

"通往诺贝尔奖的堂皇道路，通常是由简化论和还原论的思维方式取得的，这就造成了科学上越来越多的碎裂片，而真实的世界要求我们用更加整体的眼光去看问题。"

SFI认为，从局部到整体必然会导致问题的复杂化，他们将未来科学命名为"复杂科学"。

科学研究离不开数学。伽里略有句名言：宇宙这本大书是用数学语言写成的。数学是科学的世界语。数学是协助科学探索的有力工具。缺乏数学思维指导的科学活动必然是盲目的、肤浅的。

众所周知，近代科学诞生于有着雄厚数学基础的17世纪。17世纪被誉为数学史上"天才的世纪"。这个世纪取得了三项伟大的数学成就：对数方法将困难的乘除运算化归为简易的加减运算；解析几何方法将玄奥的几何命题化归为浅显的代数命题；微积分方法用简单的代数多项式逼近一般的复杂函数。总之，数学思维的基本特征是将复杂转化为简单。

数学的目的是追求简单，然而当今的科学却片面地强调"复杂"，炫耀"复杂"。复杂与简单果真是不可调和吗？数学与当今科学能够相互沟通交融吗？

新科学需要新观念

什么是"复杂"？在某种意义上，复杂意味着知识的缺乏。一个命题，在没有解决之前是复杂的，解决之后它就变得简单了；一种规律，在没有掌握之前是玄奥的，掌握之后就变得浅显了。

我们正处在计算机时代。计算机的广泛应用日益改变着世界的面貌，也深刻地影响着人们的思维方式。计算机的工作原理是简单的，它只会做加减乘除的二进制运算，然而依赖各式各样的算法，它却能承担极其复杂的计算任务。计算机上的算法究竟是怎样设计的呢？

本书以大量的算法案例透析出一个发人深省的事实：算法设计的基本理念是通过简单的重复生成复杂，或者说，将复杂转化为简单的重复。在算法设计过程中，重复就是力量。这里所说的"重复"本质上是某种演化过程。

我们深信，复杂的事物可能具有简单的演化机制，而简单的模型则可能具有复杂的演化形态。如果发展到极致，复杂与简单就可能合于一体：极端复杂就可能等于极端简单。

复杂和简单是相互变通的。它们两者是矛盾的统一体。

新观念仰赖大智慧

所谓"大智慧"，是那些历经数千年考验的人类智慧。

在2500年前，古希腊哲学家Zeno提出一个耸人听闻的命题：一个人不管跑得多快，也永远追不上爬在他前面的一只乌龟。这个"人龟追赶问题"就是所谓的Zeno悖论，它的提出触发了人类历史上"第一次数学危机"。

表面上看Zeno悖论极端荒谬，其实它同时又是极端深刻。本书揭露出一个奇妙的事实，剖析Zeno悖论所归纳出的几种算法设计技术，可用来设计出大量的计算机算法。

纵观数值算法学的发展史，中华先贤做出过杰出的贡献。祖冲之的圆周率计算曾千年称雄于世界。刘徽的千年古术至今仍熠熠生辉，指引人们探索新的计算领域。中华民族是个智慧的民族。算法设计是中华数学的强项。科学探索要国际化，首先要民族化。为要同国际先进水平接轨，先要向智慧的中华先贤讨教。在中华民族正和平崛起的今天，我们肩负着伟大而神圣的历史使命：复兴先贤伟业，重振中华雄风！

王能超

2004年9月18日

《周易》论“简易”

算法设计的基本思想是简朴的。算法设计的基本技术是简单的。算法设计追求简易。

追求简易是中华传统文化的一个重要特色。关于“简易”，我国古代经典《周易》有如下精辟的论述：

易则易知，简则易从。

易知则有亲，易从则有功。

有亲则可久，有功则可大。

可久则贤人之德，可大则贤人之业。

解释这番话的含义。

何谓“简易”？“易”，是指所讲的道理要易于理解；“简”，是指所教的方法要易于掌握。

道理易于理解就会使人亲近，彼此亲近就会持久；方法易于掌握才能收到功效，讲究功效就能壮大。

因此，追求简易是科学工作者的一项重要品德，具备这一品德才能成就伟大的事业。

前　言

自1978年以来，作者在高等教育出版社多次出版有关计算方法（数值分析）的教材，其中包括：

[1] 计算方法（1978年）

[2] 数值分析简明教程（1984年；2003年第二版）

[3] 计算方法简明教程（2003年）

这项工作是从1977年开始的。这一年恢复了高考，高等院校又焕发出生机。这一年的年底，我受命编写计算方法的“统编教材”。当时困难很多：给的学时少，仅仅提供22学时；编写时间短，要求在半年之内完成。压力变成了动力，一份结构紧凑、内容简约的教材如期“逼”了出来。

次年（1978年）5月中旬在上海召开了这份教材的审稿会，由上海交通大学孙增光教授主审，参加评审的有清华大学孙念增教授、西安交通大学游兆永教授等知名学者。与会专家对教材给予了充分的肯定。会后不久教材[1]就面世了。

为充实教材[1]，1984年又出版了教材[2]。该书以泰勒展开作为主线，被同行们评价为“泰勒公式包打天下”。该书荣获原国家教委优秀教材二等奖。

纵观形形色色的众多算法，其设计机理均可概括为“简单的重复生成复杂”。基于这一理念又编写出教材[3]。这份教材回避了泰勒展开方法，代之以几种简约的算法设计技术。内容更为简明，方法更易掌握。该书基于这些技术统一了众多常用算法，并自然地跨越到高效算法设计的学科前沿。

人类已进入计算机时代，计算机的广泛应用迫切要求普及有关计算方法的基本知识。编写本书的目的是为了进一步适应形势发展的需要，同时作者也希望为自己20余年计算方法（数值分析）的教材探索做个小结。

近两三年来作者明显地加快了教材编写工作的进度，这主要归功于高等教育出版社领导和有关编辑同志的鼎力支持，作者对此表示衷心的感谢！在此还要感谢鲁晓磊同志协助编写了篇末的附录MATLAB文件。

“谁言寸草心，报得三春晖。”作者谨将本书献给导师谷超豪教授，感谢他多年的培养、教育和关怀！

王能超

2004年8月28日

目　录

引　论

0.1 算法重在设计

0.1.1 计算机开创了现代科学的新时代

纵观上下数千年的科学史，科学的发展大致经历了古代科学、近代科学和现代科学三个历史阶段.

在遥远的古代，虽然人们在长期的社会实践中积累了不少知识，但这些知识是零碎的、不系统的和没有经过严格论证的. 古人所获取的知识大都表现为经验性的总结或猜测性的思辨，其研究方法实际上是不科学的. 在这个意义上，古代科学只是科学的萌芽，还不是真正的科学.

近代科学蓬勃兴起于 17 世纪，其奠基工作从 Galileo(伽利略，1564—1642) 开始，而由 Newton(牛顿，1642—1727)所完成. 近代科学方法强调实验和理论的紧密结合，即以实验的事实(数据和资料)为依据，通过严密的论证(数学推理)形成系统的理论. 这种科学方法促进了科学的繁荣与发展.

电子计算机的问世开创了现代科学的新时代. 随着计算机的广泛应用，科学计算正逐步上升为一种新的科学方法，它与科学实验、科学理论并列，构成科学方法论的三大组成部分.

在今天，随着科学技术革命的蓬勃发展，实际课题的规模空前扩大，所谓大型乃至超大型科学计算日益为人们所重视. 与此相适应，巨型计算机在科学计算中正扮演着越来越重要的角色. 计算机的更新换代强有力地推动着算法研究的深入，科学计算正处于蓬勃发展的新时代.

计算机是一种功能很强的计算工具. 现代超级计算机的运算速度已高达每秒万亿次. 计算机运算速度如此之快，是否意味着计算机上的算法可以随意选择呢?

举个简单的例子.

众所周知，行列式解法的 Cramer 法则原则上可用来求解线性方程组. 用这种方法求解一个 n 阶方程组，要计算 $n+1$ 个 n 阶行列式的值，总共要做 $(n+1)n!(n-1)$ 次乘除操作. 当 n 充分大时，这个计算量是相当惊人的. 譬如一个 20 阶不算太大的方程组，大约要做 10^{21} 次乘除操作. 这项计算即使用每秒

3千亿次的巨型计算机来承担,也得要连续工作

$$\frac{10^{21}}{3\times10^{11}\times60\times60\times24\times365}\approx100\ (\text{年})$$

才能完成.当然这是完全没有实际意义的.

其实,求解线性方程组有许多实用解法(参看本书第五章与第六章).譬如,运用人们熟悉的消元技术,一个20阶的线性方程组即使用普通的计算器也能很快地解出来.这个简单的案例说明,能否合理地选择算法是科学计算成败的关键.

随着计算机的广泛应用与日益普及,算法设计的重要性正越来越为人们所认识.《计算机大百科全书》在其"算法学"词条中指出:"凡与计算机打交道,无不研究各种类型的算法.""算法学是计算机科学最重要的内容,有的计算机学者甚至称,计算机科学就是算法的科学."

在知识"大爆炸"的今天,算法的数量也正以大爆炸的速度与日俱增,所涉及的文献著作数以千万计,形成浩繁的卷帙.面对这知识的汪洋大海,如何才能进行有效的学习呢?许多有志于从事科学计算的青年科学工作者正为这门学科的知识庞杂所困扰.

1976年,英国著名数学家Atiyah在他就任伦敦数学学会主席时,发表了题为"数学的统一性"的演讲①.在这次演讲中,他突出地强调了数学的简单性和统一性.他说:"数学的目的,就是用简单而基本的词汇去尽可能多地解释世界.……如果我们积累起来的经验要一代一代传下去的话,我们就必须不断地努力把它们加以简化和统一."

算法设计追求简单和统一.后文将基于一个有趣的范例,提炼出算法设计的一条基本原理,进而概括出算法设计的几种基本技术.

0.1.2 Zeno悖论的启示

古希腊哲学家Zeno在两千多年前提出过一个耸人听闻的命题:一个人不管跑得多快,也永远追不上爬在他前面的一只乌龟.这就是著名的**Zeno悖论**.

Zeno在论证这个命题时采取了如下形式的逻辑推理:设人与龟同时同向起跑,如果龟不动,那么人经过某个时刻便能赶上它;但实际上在这段时间内龟又爬行了一段路程,从而人又得重新追赶,如图0.1所示.这样每追赶一步所归结出的是同样类型的追赶问题,因而这种追赶过程永远不会终结.Zeno则据此断言人追上龟是"永远"不可能的.

Zeno悖论的提出在古希腊学术界掀起轩然大波.在Zeno悖论面前,古代的

① M Atiyah.数学的统一性.数学译林,1980年第1期

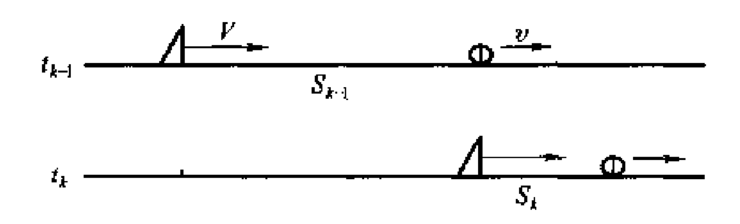


图 0.1 人龟追赶过程

数学逻辑显得无能为力，提供不出有力的论据给予驳斥，从而导致了人类文明史上"第一次数学危机".

耐人寻味的是，尽管 Zeno 悖论的论断极其荒谬，但从算法设计的角度来看它却是极为精辟的.

Zeno 悖论将人龟追赶问题表达为一连串追赶步的逐步逼近过程. 设人与龟的速度分别为 V 与 v，记 S_k 表示逼近过程的第 k 步人与龟的间距，另以 t_k 表示相应的时间，相邻两步的时间差 $\Delta t_k = t_k - t_{k-1}$. Zeno 悖论把人与龟的追赶问题分解为一追一赶两个过程(参看图 0.1)：

追的过程 先令龟不动，计算人追上龟所费的时间

$$\Delta t_k = \frac{S_{k-1}}{V} \tag{1}$$

赶的过程 再令人不动，计算龟在这段时间内爬行的路程

$$S_k = v\Delta t_k \tag{2}$$

无论是追的过程还是赶的过程，它们都是简单的行程计算. 通过这两项计算加工得出的虽然同样是追赶问题，但问题的"规模"已被大大地压缩了. 譬如，设以人与龟的间距 S_k 定义为追赶问题的**规模**，那么，经过上述两项运算手续加工后，问题的规模被压缩了 v/V 倍：

$$S_k = \frac{v}{V}S_{k-1}$$

由于龟的速度 v 远远小于人的速度 V，压缩系数 v/V 很小，因而这项计算的逼近效果极为显著. 实际上，设 $S_0 = S$ 为已知，令 $t_0 = 0$(即从人龟起跑开始计时)，则按上述手续做不了几步，追赶问题的规模 S_k 就可以忽略不计，从而得出人追上龟实际所花费的时间 t_k. 这一算法

$$\begin{cases} S_k = \dfrac{v}{V}S_{k-1}, \quad k = 1,2,\cdots \\ S_0 = S \end{cases} \tag{3}$$

可称为 **Zeno 算法**，它是 Zeno 悖论的算法描述.

上述追的过程(1)和赶的过程(2)都是简单的行程计算，Zeno 算法(3)的设

计思想是,将人与龟的追赶计算化归为简单的行程计算的重复.

总之,上述Zeno算法的每一步,都是将原先的追赶问题加工成同样类型的追赶问题,但加工后的追赶问题,其规模(如人与龟的间距,参看图0.1)已被大大地压缩了.这样,当规模变得足够小时,即可认为人已追上了龟,从而求得问题的解——人追上龟实际花费的时间.

这种反复缩减问题规模的设计策略称为**规模缩减技术**,简称**缩减技术**.缩减技术是一种基本的算法设计技术.现在介绍这种技术在直接法设计中的应用.

0.2 直接法的缩减技术

所谓直接法是这样一类算法,它通过有限步计算可以直接得出问题的精确解(如果不考虑舍入误差的话).

0.2.1 数列求和的累加算法

下述数列求和问题是人们所熟知的:

$$S = a_0 + a_1 + \cdots + a_n \tag{4}$$

这个计算模型有两个简单的特例.当 $n=0$ 即为一项和式 $S=a_0$ 时,所给计算模型就是它的解,这时不需要做任何计算.这表明,对于数列求和问题,它的解是计算模型**退化**的情形.又当 $n=1$ 即计算两项和式 $S=a_0+a_1$ 时,计算过程是**平凡**的,这时不存在算法设计问题.

现在基于这两种简单情形考察所给和式(4)的累加求和算法.设 b_k 表示前 $k+1$ 项的部分和 $a_0+a_1+\cdots+a_k$,则有

$$\begin{cases} b_0 = a_0 \\ b_k = b_{k-1} + a_k, \quad k=1,2,\cdots,n \end{cases} \tag{5}$$

而计算结果 b_n 即为所求的和值 S:

$$S = b_n \tag{6}$$

上述数列求和的累加算法,其设计思想是将多项求和(4)化归为两项求和(5)的重复.而依式(5)重复加工若干次,最终即可将所给和式(4)加工成一项和式(6)的退化情形,从而得出和值 S.

由剖析计算模型自身的演变过程.按式(5)每加工一次,所给和式(4)便减少一项,而所生成的计算模型依然是数列求和.反复施行这种加工手续,计算模型不断变形为

$$\underset{(\text{计算模型})}{n+1\text{ 项和式}} \to n\text{ 项和式} \to n-1\text{ 项和式} \to \cdots \to \underset{(\text{所求结果})}{1\text{ 项和式}}$$

这里符号“⇒”表示重复施行两项求和的加工手续.

这样,如果定义和式的项数为数列求和问题的**规模**,则所求和值可以视为规模为1的退化情形.因之,只要令和式的规模(项数)逐次减1,最终当规模为1时即可直接得出所求的值.这样设计出的算法就是累加求和算法(5).

上述累加求和算法可以视为规模缩减技术的一个范例.

0.2.2 缩减技术的设计思想

许多数值计算问题,可以引进某个实数——所谓问题的**规模**——来刻画其“大小”,而问题的解则是其规模为足够小的退化情形.求解这类问题,一种行之有效的办法是通过某种简单的运算手续逐步缩减问题的规模,直到加工得出所求的解.算法设计的这种技术称为**规模缩减技术**,简称**缩减技术**.

缩减技术所适用的一类问题是,求解这类问题的困难所在是它的规模(适当定义)比较大.针对这类问题运用缩减技术,就是设法逐步缩减计算问题的规模,直到规模变得足够小时直接生成或方便地求出问题的解.

缩减技术的设计思想可用**大事化小,小事化了**这句俗话来概括.

所谓“大事化小”意即逐步压缩问题的规模.在运用缩减技术时,“大事”是如何“化小”的呢?这个处理过程具有如下两项基本特征:

(1) **结构递归**

大事化小是逐步完成的,其每一步将所考察的计算模型加工成同样类型的计算模型,因而这类算法具有明晰的递归结构.

(2) **规模递减**

每一步加工前后的计算模型虽然从属于同一类型,但其规模已被压缩了.压缩系数愈小则算法的效率愈高.

再考察“小事化了”的处理过程.所谓“小事化了”,是指当问题的规模变得足够小时即可直接或方便地得出问题的解.

“小事”是如何“化了”的呢?

对于某些计算模型,如前面讨论过的数列求和问题,它们的规模为正整数,而其解则是规模为0或1的退化情形.这时只要设法使规模逐次减1,加工若干步后即可直接得出所求的解.这里小事化了是直截了当的.

这样设计出的一类算法统称**直接法**.前述数列求和的累加算法以及下述多项式求值的秦九韶算法都是直接法.

0.2.3 多项式求值的秦九韶算法

微积分方法的核心是逼近法.多项式是微积分学中最为基本的一种逼近工具,因而多项式求值算法在微积分计算中具有重要意义.

设要对给定的 x 计算下列多项式的值：

$$P = a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n = \sum_{k=0}^{n} a_k x^{n-k} \tag{7}$$

由于计算每一项 a_kx^{n-k} 需做 $n-k$ 次乘法，如果先逐项计算 a_kx^{n-k}，然后再累加求和计算多项式的值 P，这种**逐项生成算法**所要耗费的乘法次数为

$$Q = \sum_{k=0}^{n}(n-k) \approx \frac{n^2}{2}$$

当 n 充分大时这个计算量是相当大的.

现在设法改进这一算法.类似于数列求和计算，首先考察两个特例：当 $n=0$ 时所给计算模型即为所求的解

$$P = a_0$$

这时不需要做任何计算.又当 $n=1$ 时计算模型

$$P = a_0x + a_1$$

为简单的一次式，这时虽然需要进行计算，但不存在算法设计问题.

注意到当 $x=1$ 时多项式(7)便退化为和式(4)，可以类比数列求和算法的设计过程讨论多项式求值算法的设计问题.

设将多项式的次数规定为多项式求值问题的规模，如果从式(7)的前面两项中提出公因子 x^{n-1}，则有

$$P = (a_0x + a_1)x^{n-1} + \sum_{k=2}^{n} a_k x^{n-k}$$

这样，如果算出一次式

$$v_1 = a_0x + a_1$$

的值，则所给计算模型(7)便化归为 $n-1$ 次式

$$P = v_1x^{n-1} + \sum_{k=2}^{n} a_k x^{n-k}$$

的计算，从而使问题的规模减少了1次.不断地重复这种加工手续，使计算问题的规模逐次减1，则经过 n 步即可将所给多项式的次数降为0，从而获得所求的解.这样设计出的算法是算法0.1.

算法 0.1 令 $v_0 = a_0$，对 $k=1,2,\cdots,n$ 计算

$$v_k = x \cdot v_{k-1} + a_k \tag{8}$$

则结果 $P = v_n$ 即为所给多项式(7)的值.

容易看出，按递推算式(8)计算多项式(7)的值，总共只要做 n 次乘法，其计算量远比前述逐项生成算法的计算量少.这是一种优秀算法.

这一优秀算法称作**秦九韶算法**.它是我国南宋大数学家秦九韶(公元13世

纪)最先提出来的.需要提醒注意的是,国外文献常称这一算法为 **Horner 算法**,其实 Horner 的工作比秦九韶晚了五、六百年.

秦九韶算法说明,n 次式(7)的求值问题可化归为一次式(8)求值计算的重复.设以符号"⇒"表示一次式的求值手续,则秦九韶算法的模型加工流程如下:

$$\begin{matrix}n\text{ 次式求值}\\(\text{计算模型})\end{matrix}\Rightarrow n-1\text{ 次式求值}\Rightarrow n-2\text{ 次式求值}\Rightarrow\cdots\Rightarrow\begin{matrix}0\text{ 次式求值}\\(\text{计算结果})\end{matrix}$$

0.2.4 算法的框图描述

同人工手算的计算方法意义不同,将要研究的算法是为计算机提供的,因此,解题方案中的每个细节都必须加以定义,并且要完整地描述整个计算过程.这就是说,所谓计算机算法不仅仅是单纯的数学公式,而是指解题方案的准确而完整的描述.

刻画计算流程有多种方式,本书常用框图直观地显示算法的全貌.

本书将使用两种形式的框:一种是叙述框▭,计算公式就填在这种框内;另一种是检查框◇,它表示算法的判断部分.检查框有两个出口,究竟选择哪个出口,要视框内的判断条件是否满足而定.

今后所有的框图均以㊉框标志计算过程开始启动,而以㊇框表示计算过程的最终结束.另以箭头"→"指明各框执行的顺序.

现在考察秦九韶算法的计算流程.

按秦九韶算式(8)进行计算,每求出一个"新值"v_k 以后,"老值"v_{k-1} 便失去继续保存的价值,因此可将新值 v_k 存放在老值 v_{k-1} 所占用的单元内.这样,只要设置一个单元 v 进行累算,而将算式(8)表为如下动态形式:

$$v \Leftarrow x \cdot v + a_k, \quad k=1,2,\cdots,n$$

这里箭头"⇐"用以刻画赋值手续.执行这组算式之前,应先送初值 a_0 到单元 v 中:

$$v \Leftarrow a_0$$

图 0.2 描述了多项式(7)求值的秦九韶算法,其中:

框 1 为**准备部分**.单元 v 中送初值 a_0,单元 k 中送计数值 1.

框 2 为**计算部分**.每循环一次,单元 v 中的老值 v_{k-1} 为新值 v_k 所替换.

框 3 为**控制部分**.检查单元 k 中的计数值以判断循环应否结束.当计数值为 n 时输出 v 中结果,否则转框 4.

框 4 为**修改部分**.修改单元 k 中计数值,然后转框 2 再做下一步的计算.

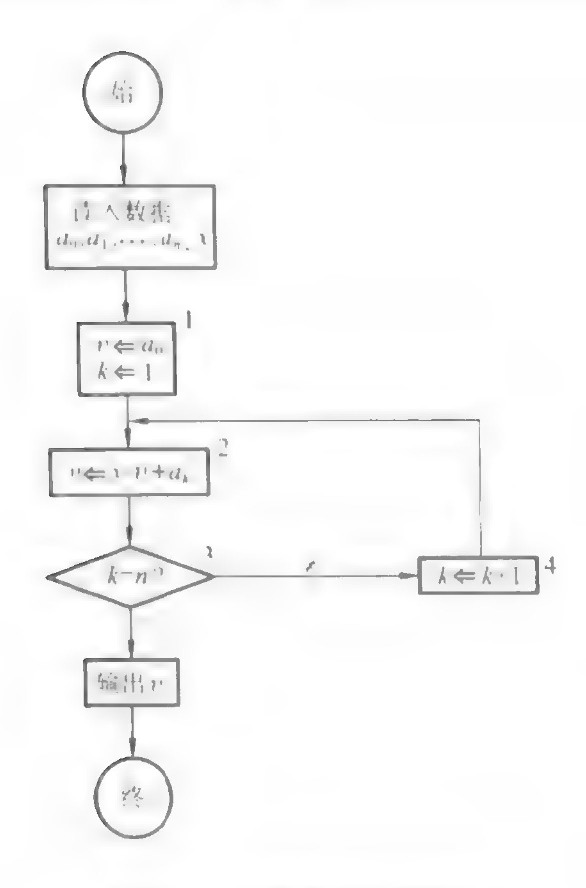


图0.2 秦九韶算法的计算流程

0.3 迭代法的校正技术

上一节介绍了设计直接法的缩减技术. 缩减技术针对这样的问题, 它的解是规模足够小(通常规模为0或1)的退化情形. 这样, 只要设法令规模逐次减1, 即可将计算模型逐步加工成解的形式. 这种加工过程可用"大事化小, 小事化了"这句俗话来概括.

有些问题的"大事化小"过程似乎无法了结. Zeno悖论强调人"永远"追不上龟正是为了突出这层含义. 这是一类无限的逼近过程, 其问题的规模通常是实数. 正如前述Zeno算法所看到的, 如果所设计出的逼近过程按某个比例常数一致地缩减, 那么, 适当提供某个精度即可控制计算过程的终止. 这样设计出的算法通常称作**迭代法**.

0.3.1 Zeno悖论中的"Zeno钟"

Zeno悖论所表述的人龟追赶问题其实是容易求解的. 设人与龟起初相距 S, 两者速度分别为 V 与 v, 则容易列出方程

$$Vt - vt = S \tag{9}$$

因之人追上龟实际所花费的时间

$$t^* = \frac{S}{V - v}$$

我们再运用所谓预报校正技术处理这个简单问题，为将来求解一般非线性方程做准备.

设有解 t^* 的某个**预报值** t_0，希望提供校正量 Δt，使**校正值**

$$t_1 = t_0 + \Delta t$$

能更好地满足所给方程(9)，即尽可能准确地成立

$$V(t_0 + \Delta t) - v(t_0 + \Delta t) \approx S$$

注意到 v 是个小量，设校正量 Δt 也是个小量，从上式当中略去高阶小量 $v\Delta t$，得

$$V(t_0 + \Delta t) - vt_0 = S \tag{10}$$

求解这个方程，所定出的校正值 $t_1 = t_0 + \Delta t$ 为

$$t_1 = \frac{S + vt_0}{V}$$

进一步视 t_1 为新的预报值重复施行上述手续求出新的校正值 t_2，依 t_2 再定出 t_3，如此反复地做下去，即可生成一个近似值序列 $t_1, t_2, \cdots$，这就规定了一个迭代过程，其迭代公式为

$$t_{k+1} = \frac{S + vt_k}{V}, \quad k = 0, 1, 2, \cdots \tag{11}$$

Zeno 悖论所表述的逼近过程正是这种迭代过程，当 $k \to \infty$ 时式(11)的迭代结果 t_k 将收敛到人追上龟所需的时间 t^*.

那么，Zeno 强调人"永远"追不上龟试图表达什么含义呢？

我们知道，任何形式的重复均可作为时间的量度. Zeno 在刻画人龟追赶过程时实际上设置了两个"时钟"：一个是日常钟 t_k，其含义无需解释；Zeno 又将迭代次数 k——即一追一赶过程(参看图 0.1)的重复次数视为另一种"时钟"，不妨称这一时钟为 **Zeno 钟**. Zeno 钟 k 采取离散的计数方式，它仅仅取正整数值. Zeno 公式(11)表明，当 Zeno 钟 k 趋于∞时人才能追上龟，Zeno 正是依据这一事实断言"人永远追不上龟".

0.3.2 开方算法

早在四千多年前，在亚洲西南部的古巴比伦地区(现今伊拉克境内)就已经萌发出数学智慧的幼芽. 古巴比伦数学取得了一系列重要成就，譬如制成了有关平方根的数表. 古巴比伦人制造开方表的方法难以考证，不过可以想象其计算方

法必定相当的简单.

现代电子计算机上又是怎样计算开方值的呢?

相对于加减乘除四则运算来说,开方运算无疑是复杂的.人们自然希望将复杂的开方运算归结为四则运算的重复,为此需要设计某种算法.

给定 $a>0$,求开方值$\sqrt{a}$的问题就是要解方程

$$x^2-a=0 \tag{12}$$

这是个非线性的二次方程,从初等数学的角度来看它的求解有难度.该如何化难为易呢?

设给定某个预报值 x_0,我们希望借助于某种简单方法确定校正量 Δx,使校正值 $x_1=x_0+\Delta x$ 能够比较准确地满足所给方程(12),即有

$$(x_0+\Delta x)^2\approx a$$

假设校正量 Δx 是个小量,为简化计算,舍弃上式中的高阶小量$(\Delta x)^2$,而令

$$x_0^2+2x_0\Delta x=a$$

这是关于 Δx 的一次方程,据此定出 Δx,从而对校正值 $x_1=x_0+\Delta x$ 有

$$x_1=\frac{1}{2}\left(x_0+\frac{a}{x_0}\right)$$

反复施行这种预报校正手续,即可导出**开方公式**

$$x_{k+1}=\frac{1}{2}\left(x_k+\frac{a}{x_k}\right),\quad k=0,1,2,\cdots \tag{13}$$

从给定的某个初值 $x_0>0$ 出发,利用上式反复迭代,即可获得满足精度要求的开方值$\sqrt{a}$.

算法 0.2 任给 $x_0>0$,对 $k=0,1,2,\cdots$ 执行算式(13),直到偏差 $|x_{k+1}-x_k|<\varepsilon$($\varepsilon$ 为给定精度)为止,最终获得的近似值 x_k 即为所求.

设置两个单元 x_0,x_1 分别存放迭代前后的近似值,则上述开方算法的计算流程如图 0.3 所示.

例 用开方算法求$\sqrt{2}$,设取 $x_0=1$.

解 $\sqrt{2}$ 的准确值为 1.414 213 56….取 $a=2$ 按式(13)迭代 4 次即可得出准确到 $\varepsilon=10^{-6}$的结果 1.414 214(见表 0.1)[①].

① 称近似值 x 关于精确值 x^*"准确到"ε,如果两者的偏差$|x-x^*|<\varepsilon$.

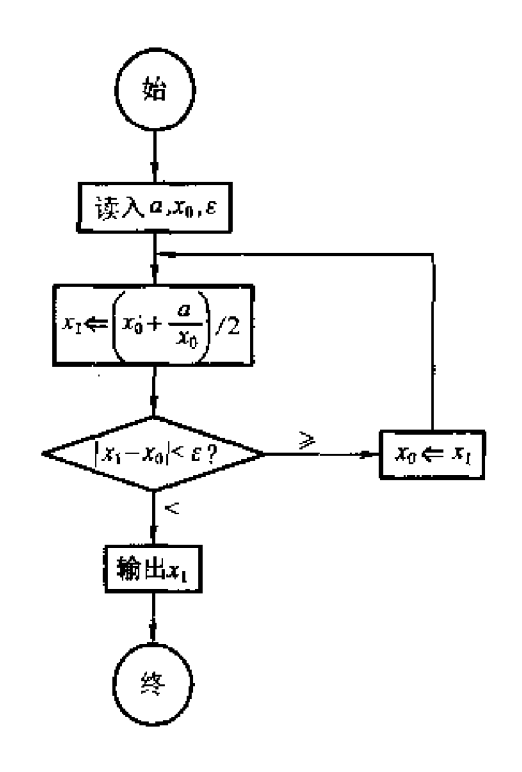


图 0.3 开方算法的计算流程

表 0.1

k	x_k	k	x_k
0	1.000 000	3	1.414 216
1	1.500 000	4	1.414 214
2	1.416 667	5	1.414 214

0.3.3 校正技术的设计思想

上述开方算法虽然结构简单,但它深刻地揭示了校正技术的设计思想.前面已指出,算法设计的基本原则是以简御繁,即将复杂计算化归为一系列简单计算的重复.迭代法突出地体现了这项原则,其设计思想可概括为**以简御繁,逐步求精**.

所谓"以简御繁",是指构造某个简化方程近似替代原先比较复杂的方程,以确定所给预报值的校正量.这种用于计算校正量的简化方程称作**校正方程**.关于校正方程有以下两项基本要求:

(1) **逼近性**

它与所给方程是近似的.逼近程度越高,所获得的校正量越准确.

(2) **简单性**

校正方程越简单,所需计算量越小.求校正量通常采取显式计算.

应当指出的是，在设计校正方程时，上述逼近性与简单性往往是彼此矢被的两个矛盾因素．逼近性高，往往会导致校正方程的复杂化，使计算量显著增加．在具体设计校正方程时需要权衡得失．

如何利用简单的校正方程获得原方程的解呢？为使简单转化为复杂，一种行之有效的途径是递推化．对于给定的某个预报值 t_0，利用校正方程计算校正量从而得出校正值 t_1，这就完成了迭代过程的一步．是否需要继续迭代取决于校正量是否满足精度要求．如果不满足精度要求，则用老的校正值充当新的预报值重复上述步骤．如此继续下去，直到所获得的校正值满足精度要求为止．由此可见，迭代过程是个"逐步求精"的递归过程．

0.4 算法优化的松弛技术

0.4.1 Zeno 算法的升华

再考察 Zeno 算法．对于给定的预报值 t_0，按式(11)校正值为

$$t_1 = \frac{S + vt_0}{V}$$

据此有

$$Vt_1 - vt_0 = S$$

两端同除以 $V - v$，有

$$\frac{V}{V-v}t_1 - \frac{v}{V-v}t_0 = \frac{S}{V-v}$$

需要提醒注意的是，上式右端

$$t^* = \frac{S}{V-v}$$

为人追上龟实际所需的时间，即人龟追赶问题的精确解（参看 0.3 节）．由此可见，精确解 t^* 等于预报值 t_0 同它的校正值 t_1 两者的加权平均：

$$t^* = (1+\omega)t_1 - \omega t_0, \quad \omega = \frac{v}{V-v} \tag{14}$$

我们看到，这里将任意一对粗糙的迭代值 t_0, t_1（它们的精度可能都很差）按固定方程式(14)进行松弛，结果总可以获得所给方程的精确解 t^*．这种化粗为精的加工效果是奇妙的．

0.4.2 松弛技术的设计思想

在实际计算中常常可以获得目标值 F^* 的两个相伴随的近似值 F_0 与 F_1，

如何将它们加工成更高精度的结果呢？改善精度的一种简便而有效的办法是，取两者的某种加权平均值作为改进值，即令

$$\begin{aligned}\hat{F} &= (1-\omega)F_0 + \omega F_1 \\ &= F_0 + \omega(F_1 - F_0)\end{aligned}$$

也就是说，适当选取权系数 ω 来调整校正量 $\omega(F_1 - F_0)$，以将近似值 F_0 加工成更高精度的结果 $\hat{F}$. 正是由于这种方法基于校正量的调整与松动，故称之为**松弛技术**.

有一种情况特别引人注目. 如果所提供的一对近似值 F_0, F_1 有优劣之分，譬如 F_1 为优而 F_0 为劣，这时往往采取如下松弛方式（参看式(14)）：

$$\hat{F} = (1+\omega)F_1 - \omega F_0, \quad \omega > 0$$

即在松弛过程中张扬 F_1 的优势而抑制 F_0 的劣势，这种设计策略称作**超松弛**.

概括地说，超松弛技术的设计机理是**优劣互补，化粗为精**.

值得指出的是，欲使松弛技术真正实现提高精度的效果，关键在于松弛因子 ω 的选取. 而这往往是相当困难的. 令人不可思议的是，早在两千年前，智慧的中华先贤就已经掌握这种高深精湛的算法设计技术.

0.4.3 千古绝技"割圆术"

在数学史上，圆周率这个奇妙的数字牵动着一代又一代数学家的心，不少人为之耗费了毕生的精力. 按现存文献记载，在这方面做出过突出贡献的，当首推古希腊的阿基米德，在公元前 3 世纪，他用圆内接与外切正 96 边形逼近圆周，得出 π 的近似值 3.14. 这是公元前最好的结果.

关于圆周率计算问题，我国古代数学家也做出过杰出的贡献. 魏晋大数学家刘徽（公元 3 世纪）曾提出过所谓"割圆术"，他从内接正 6 边形割到正 12 边形，再割到正 24 边形，如此一直割到内接正 3 072 边形，得出 π 的近似值 3.141 6. 这是当时最好的结果.

刘徽不但提供了一种圆周率计算的迭代算法，而且还给出一个加速逼近公式. 这是刘徽割圆术中最为精彩的部分[①].

刘徽用内接正 n 边形的面积 S_n 来逼近圆面积，并取半径 $r=10$ 进行实际计算. 他发现，利用 $S_{192} = 314\frac{64}{625}$ 与 $S_{96} = 313\frac{584}{625}$ 两个粗糙的数据进行松弛，即可获得高精度的值 S_{3072}. 刘徽提供了如下加工过程：

① 王能超著. 千古绝技"割圆术". 第二版. 武汉：华中科技大学出版社，2003

$$\hat{S}=S_{192}+\frac{36}{105}(S_{192}-S_{96})=314\,\frac{64}{625}+\frac{36}{105}\times\frac{105}{625}$$
$$=314\,\frac{100}{625}=314\,\frac{4}{25}\approx S_{3\,072}$$

据此获知 $\pi=3.141\,6$.

就这样,刘徽利用两个粗糙的近似值 S_{96},S_{192} 进行松弛,结果获得了高精度的近似值 $S_{3\,072}$,从而实现了圆周率计算的一次革命性飞跃.

刘徽的"割圆术"在数学史上占有重要的地位,它开创了加速算法设计的先河.直到 1 700 多年后的 20 世纪,西方数学家才基于所谓余项展开式探讨逼近过程的加速问题(参看本书第二章 2.5 节).

松弛技术是一种加速技术,而加速算法则是一类高效算法.高效算法的设计将算法学提升到新的高度,成为当代高性能计算中众所瞩目的亮点.

小 结

学习计算机上的数值算法,要领悟一条基本原理,区分两类基本方法,掌握三种基本技术.

计算机上的算法形形色色,变化万千,但万变不离其宗.不管哪一种计算机算法,其设计机理都是将复杂化归为简单的重复.或者说,通过简单的重复生成复杂.在算法设计与算法实现过程中,重复就是力量!

计算机上的数值算法大致分为直接法与迭代法两大类.直接法通过有限步计算直接得出问题的解,而迭代法则通过某种迭代过程逐步逼近所求的解.本书末尾两章介绍求解线性方程组的直接法与迭代法,将会看到,迭代法与直接法既彼此对立又相互统一.

数值算法的设计技术大致有三种:化大为小的缩减技术,化难为易的校正技术以及化粗为精的松弛技术.缩减技术与校正技术分别适用于直接法与迭代法的设计,而恰当地运用松弛技术有可能显著地提高迭代过程的收敛速度.

随着电子计算机的广泛应用,人们已经就各种数值问题提出了大量的算法.所涉及的文献著作数以千万计,形成浩繁的卷帙.面对这知识的汪洋大海,该如何进行有效的学习和研究呢?

学习和研究算法,应当从最简单的做起.

每学一个专题,首先剖析一两个最简单、最初等的范例.譬如前述求和公式与开方公式.前已看到,基于这些极其简单的范例可提炼出一般性的设计技术,这是个"点石成金"的过程.

要学好算法,关键在于将各种各样的具体算法进行归纳分类,并触类旁通.

《周髀算经》中上古先贤陈子告诫后人，学习算法要有“智类之明”，“问一类而以万事达”，这是一服解读各种算法的灵丹妙药．

习　题　0

1．用缩减技术计算

$$T = \prod_{i=0}^{n} a_i$$

设计出累乘求积算法，并绘制计算流程图．

2．运用缩减技术设计

$$Q = \sum_{i=0}^{n} \left(\prod_{j=0}^{i-1} b_j \right) a_i$$

的求值算法（这里约定 $\prod_{j=0}^{-1} b_j = 1$），并绘制计算流程图．

3．设 $P(x) = a_0 x^n + a_1 x^{n-1} + \cdots + a_n$，试对给定 x 设计导数 $P'(x)$ 的求值算法．

4．用校正技术解方程 $\frac{1}{x} - a = 0$，设计求倒数 $1/a$ 而不用除法的迭代算法，并绘制计算流程图．

5．取 $x_0 = 1$，用迭代公式 $x_{k+1} = \frac{1}{1 + x_k}$ 计算方程 $x^2 + x - 1 = 0$ 的正根 $x^* = \frac{-1 + \sqrt{5}}{2}$，要求精度 10^{-5}．

6．将题 5 迭代前后的值加权平均生成迭代公式

$$x_{k+1} = \omega x_k + (1 - \omega) \frac{1}{1 + x_k}$$

试验证，若取 $\omega = \frac{7}{25}$ 则上列公式可改进题 5 的收敛速度．

第一章 插值方法

1.1 插值平均

人类在长期的生产实践和科学探索过程中获得了大量的数据，制成了各种各样的数据表．所谓"插值"，通俗地说，就是在所给数据表中再"插"进一些所需要的值．

插值方法源于科学研究的实践．在17世纪，西欧科学探索活动空前活跃，发生了诸如哥伦布发现"新大陆"、麦哲伦环球航行等一系列重大事件．科学实践的客观需要强烈地推动了插值方法的深入研究．

插值方法是一类古老的数学方法．早在一千多年前的隋唐时期，智慧的中华先贤在制定历法的过程中就已经广泛地运用了插值技术．公元6世纪，隋朝刘焯已将等距节点的二次插值应用于天文计算，中国人关于插值方法的研究早于西方一千多年．

1.1.1 问题的提出

工程实践与科学研究过程中会碰到各式各样的函数 $f(x)$，有的表达式很复杂，有的甚至提供不出 $f(x)$ 的表达式，而只是通过实验和计算获得若干节点 x_i 上的函数值 $f(x_i)=y_i$，或者说，只是提供了一张**函数表**（如表1.1所示）．

表 1.1

x	x_0	x_1	x_2	$\cdots$	x_n
$y=f(x)$	y_0	y_1	y_2	$\cdots$	y_n

所谓**插值**，就是设法利用已给数据表求出给定点 x 的函数值 y．表中的数据点 $x_i(i=0,1,\cdots,n)$ 称**插值节点**，所要插值的点 x 称**插值点**．

插值计算的目的在于，通过尽可能简便的方法，利用所给函数值 y_i 加工出插值点 x 上具有足够精度的插值结果 y．

在这种意义上，插值过程是个数据加工的过程．

不言而喻，所给函数表1.1的每个数据 y_i 均可近似地充当插值结果 y，只是

精度不够.自然会想到,能否将这些数据适当组合生成所要的插值结果呢?

问题在于,如何选取组合系数 λ_i,使组合值

$$y = \sum_{i=0}^{n} \lambda_i y_i \tag{1}$$

具有"尽可能高"的精度.为此要求提供一种判别插值精度高低的准则.

1.1.2 代数精度的概念

再考察插值公式(1).注意到 $y_i = f(x_i)$ 而所求结果 y 为 $f(x)$ 的近似值,对应于插值公式(1)有如下近似关系式

$$f(x) \approx \sum_{i=0}^{n} \lambda_i f(x_i) \tag{2}$$

这样,插值方法所要解决的问题是,如何选取系数 λ_i 使近似关系式(2)对于所给函数 $f(x)$ 能够比较准确地成立.

微积分的 Taylor 分析表明,一般函数可用代数多项式来近似地刻画.因之,为要保证函数关系式(2)具有"尽可能高"的精度,只要令它对于"尽可能多"的代数多项式 $f(x)$ 均能准确地成立.具体地说:

定义 1 称近似关系式(2)具有 m **阶精度**,如果它对于次数 $\leqslant m$ 的多项式均能准确成立,或者说,它对于幂函数 $f=1,x,x^2,\cdots,x^m$ 均能准确成立,而对于 $f=x^{m+1}$ 不准确.

称插值公式(1)具有 m 阶精度,如果其对应的近似关系式(2)具有 m 阶精度.

特别地,令式(2)对于零次式 $f=1$ 准确成立,可列出方程

$$\sum_{i=0}^{n} \lambda_i = 1$$

这表明插值方法是一种平均化方法.因此这项数据加工手续亦可称作**插值平均**.

下一节将会看到,基于定义 1 的精度判别准则,插值公式的构造问题可化归为求解代数方程组的代数问题.

1.2 Lagrange 插值公式

先考察简单的两点插值.

1.2.1 两点插值

设给定含有两个节点的数据表(如表 1.2 所示)

表 1.2

x	x_0	x_1
$y=f(x)$	y_0	y_1

求作形如

$$y=\lambda_0 y_0+\lambda_1 y_1 \tag{3}$$

的插值公式，式中 λ_0,λ_1 为待定系数. 为此令其对应的近似关系式

$$f(x)\approx\lambda_0 f(x_0)+\lambda_1 f(x_1)$$

具有一阶精度，即对于幂函数 $f=1, f=x$ 准确成立，则可列出方程组

$$\begin{cases}\lambda_0+\lambda_1=1\\ \lambda_0 x_0+\lambda_1 x_1=x\end{cases}$$

运用 Cramer 法则解得

$$\lambda_0=\begin{vmatrix}1 & 1\\ x & x_1\end{vmatrix}\Bigg/\begin{vmatrix}1 & 1\\ x_0 & x_1\end{vmatrix}=\frac{x-x_1}{x_0-x_1}$$

同理有

$$\lambda_1=\frac{x-x_0}{x_1-x_0}$$

从而依数据表 1.2 作出的形如(3)的插值公式是

$$y=\frac{x-x_1}{x_0-x_1}y_0+\frac{x-x_0}{x_1-x_0}y_1 \tag{4}$$

这就解决了下述两点插值问题.

问题 1　设给定数据表 1.2，求作形如(3)的插值公式使具有一阶精度.

例 1　运用引论 0.3.2 小节的开方算法可造出开方表 1.3，试利用这张数据表依两点插值公式(4)求开方值 $\sqrt{3.45}$.

表 1.3

x	1	2	3	4	5	6	7	8
$y=\sqrt{x}$	1.000 00	1.414 21	1.732 05	2.000 00	2.236 07	2.449 49	2.645 75	2.828 43

解　取靠近插值点 $x=3.45$ 的两个插值节点 $x_0=3, x_1=4$ 依式(4)求得 $y=1.852\ 63$，与准确值 $\sqrt{3.45}=1.857\ 417\ 56\cdots$ 比较，这一结果有 3 位有效数字①.

① 称某一近似值有 m 位有效数字，如果它与准确值的误差不超过其 m 位的半个单位. 例 1 的计算结果 1.852 63 与准确值比较，其误差小于 $\frac{1}{2}\times10^{-2}$，即不超过第三位的半个单位，故它有 3 位有效数字.

1.2.2 三点插值

两点插值的精度不高.为改善精度进而考察三点插值.

问题 2 设给定含有三个节点的数据表(如表 1.4 所示)

表 1.4

x	x_0	x_1	x_2
$y=f(x)$	y_0	y_1	y_2

求作插值公式

$$y=\lambda_0 y_0+\lambda_1 y_1+\lambda_2 y_2 \tag{5}$$

使具有三阶精度.

解 按定义,为使式(5)具有三阶精度,要求其对应的近似关系式

$$f(x)\approx\lambda_0 f(x_0)+\lambda_1 f(x_1)+\lambda_2 f(x_2)$$

对于 $f=1,x,x^2,x^3$ 准确成立,据此列出方程组

$$\begin{cases}\lambda_0+\lambda_1+\lambda_2=1\\ \lambda_0 x_0+\lambda_1 x_1+\lambda_2 x_2=x\\ \lambda_0 x_0^2+\lambda_1 x_1^2+\lambda_2 x_2^2=x^2\end{cases} \tag{6}$$

对于给定的插值点 x,上式是关于待定参数 $\lambda_0,\lambda_1,\lambda_2$ 的线性方程组.仍用 Cramer 法则求解.为此考察其系数行列式

$$V(x_0,x_1,x_2)=\begin{vmatrix}1 & 1 & 1\\ x_0 & x_1 & x_2\\ x_0^2 & x_1^2 & x_2^2\end{vmatrix}$$

注意到

$$V(x_0,x_1,t)=\begin{vmatrix}1 & 1 & 1\\ x_0 & x_1 & t\\ x_0^2 & x_1^2 & t^2\end{vmatrix}$$

关于变元 t 是个二次式,它有两个零点 x_0,x_1,且其首项(t^2 项)的系数为

$$\begin{vmatrix}1 & 1\\ x_0 & x_1\end{vmatrix}=x_1-x_0$$

因而可以断定

$$V(x_0,x_1,t)=(x_1-x_0)(t-x_0)(t-x_1)$$

从而有

$$V(x_0,x_1,x_2)=(x_1-x_0)(x_2-x_0)(x_2-x_1)$$

于是方程组(6)的解

$$\lambda_0 = \frac{V(x, x_1, x_2)}{V(x_0, x_1, x_2)} = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}$$

考虑到节点 x_0, x_1, x_2 地位对等,又有

$$\lambda_1 = \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}$$

$$\lambda_2 = \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}$$

故问题 2 所求的插值公式为

$$y = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}y_0 + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}y_1 + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}y_2 \tag{7}$$

附带指出,当节点 x_0, x_1, x_2 互异时系数行列式 $V(x_0, x_1, x_2)$的值异于0,因而问题 2 的解存在且唯一,即这时插值公式仅有式(7)一种.

例 2 依据数据表 1.3 利用三点插值公式(7)计算开方值 $\sqrt{3.45}$.

解 注意到靠近插值点 $x = 3.45$ 的三个节点 $x_0 = 2, x_1 = 3, x_2 = 4$,利用插值公式(7)求得 $y = 1.858\ 80$,同准确值比较,这一结果有 3 位有效数字,其误差小于例 1 的两点插值.

1.2.3 多点插值

问题 3 对于给定的数据表 1.1 即(x_i, y_i), $i = 0, 1, \cdots, n$,求作形如(1)即

$$y = \sum_{i=0}^{n} \lambda_i y_i$$

的插值公式,使具有 n 阶精度.

仿照前面的讨论,有下述论断:

定理 问题 3 唯一可解,且其解具有形式

$$y = \sum_{i=0}^{n} \left(\prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \right) y_i \tag{8}$$

这一公式称作 **Lagrange 插值公式**. 式(4)与式(7)是式(8)取 $n = 1, 2$ 的特殊情形.

证 事实上,为使求积公式(1)有 n 阶精度,要求其对应的近似关系式

$$f(x) \approx \sum_{i=0}^{n} \lambda_i f(x_i)$$

对于 $f = 1, x, x^2, \cdots, x^n$ 均准确成立,据此列出方程组

$$\begin{cases} \lambda_0+\lambda_1+\lambda_2+\cdots+\lambda_n=1 \\ \lambda_0 x_0+\lambda_1 x_1+\lambda_2 x_2+\cdots+\lambda_n x_n=x \\ \lambda_0 x_0^2+\lambda_1 x_1^2+\lambda_2 x_2^2+\cdots+\lambda_n x_n^2=x^2 \\ \cdots\ \cdots\ \cdots\ \cdots \\ \lambda_0 x_0^n+\lambda_1 x_1^n+\lambda_2 x_2^n+\cdots+\lambda_n x_n^n=x^n \end{cases}$$

其系数行列式是所谓 **Vandermonde 行列式**

$$V(x_0,x_1,x_2,\cdots,x_n)=\begin{vmatrix} 1 & 1 & 1 & \cdots & 1 \\ x_0 & x_1 & x_2 & \cdots & x_n \\ x_0^2 & x_1^2 & x_2^2 & \cdots & x_n^2 \\ \cdots & \cdots & \cdots & \cdots & \\ x_0^n & x_1^n & x_2^n & \cdots & x_n^n \end{vmatrix}$$

仿照 1.2.2 节的做法不难证明

$$V(x_0,x_1,x_2,\cdots,x_n)=(x_1-x_0)(x_2-x_0)(x_2-x_1)\cdots\prod_{j=0}^{n-1}(x_n-x_j)$$

从而运用 Cramer 法则求得

$$\lambda_0=\frac{V(x,x_1,\cdots,x_n)}{V(x_0,x_1,\cdots,x_n)}=\prod_{j=1}^{n}\frac{x-x_j}{x_0-x_j}$$

考虑到节点 $x_i(i=0,1,\cdots,n)$地位对等,知

$$\lambda_i=\prod_{\substack{j=0\\ j\neq i}}^{n}\frac{x-x_j}{x_i-x_j},\quad i=0,1,\cdots,n$$

故有插值公式(8).定理得证.

Lagrange 公式(8)具有累乘累加的嵌套结构,容易编制其计算程序.事实上,式(8)在逻辑上表现为二重循环,内循环(j 循环)累乘求得系数 λ_i,然后再通过外循环(i 循环)累加得出插值结果 y.

算法 1.1(Lagrange 插值) 设给定数据表$(x_i,y_i),i=0,1,\cdots,n$ 及插值点x,据式(8)求得插值结果 y.

图 1.1 描述了 Lagrange 插值的计算流程.

针对所给数据表运用插值方法时,人们往往以为选取的插值节点越多,插值结果越准确.这种看法其实不一定可靠.实际计算结果表明,如果选取的节点过多,Lagrange 插值的计算结果反而会严重失真.

实际运用插值方法时,通常在插值点邻近适当选取少许几个节点(两、三个或者四、五个,如例 1、例 2 那样)进行插值,这种插值方法称作**分段插值法**.

值得指出的是,我国古代天文学家在制定历法的过程中曾深入地研究过分段插值方法,做出了杰出贡献.例如,隋朝刘焯的杰作《皇极历》(600 年)、中唐僧

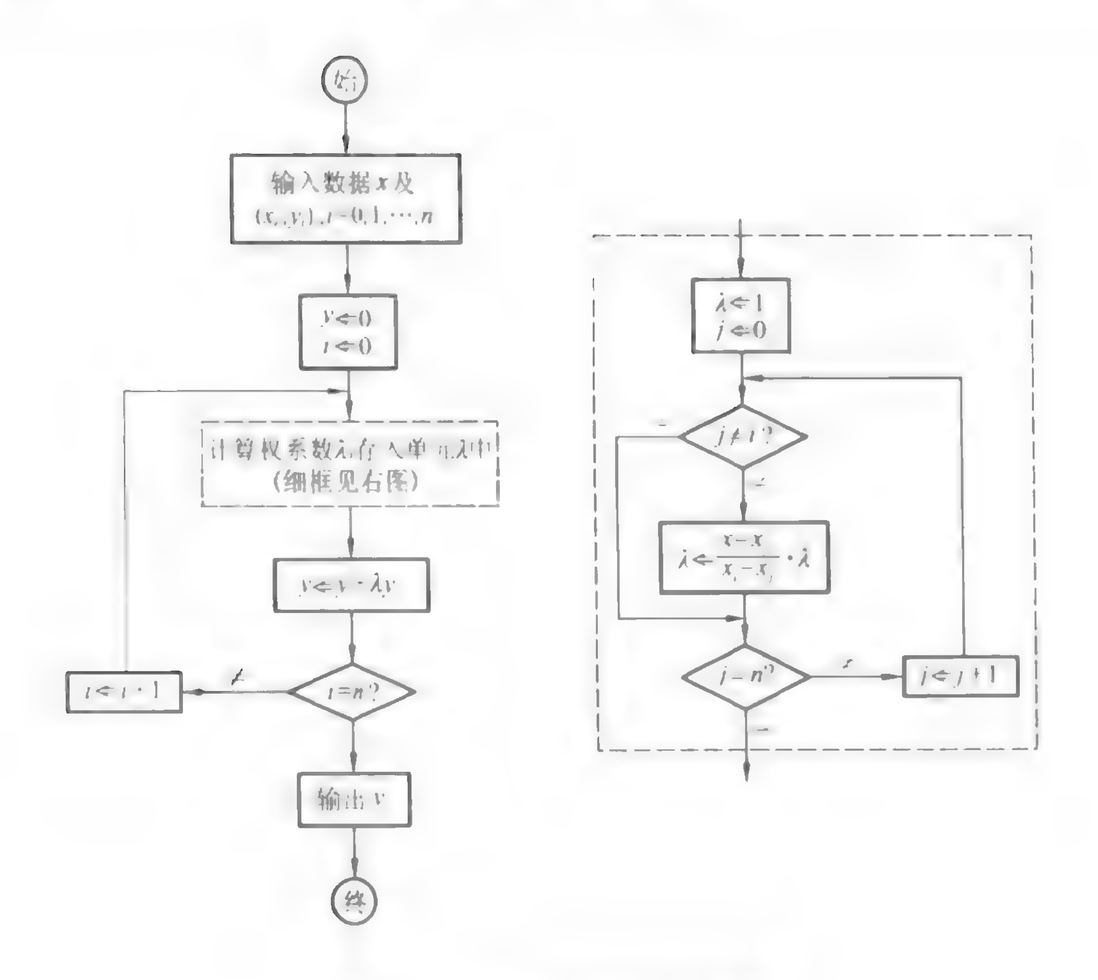


图 1.1　Lagrange 插值的计算流程

一行所造《大衍历》(727 年)都使用了分段二点插值. 特别是, 晚唐徐昂造《宣明历》(822 年)所使用的插值技术正是等距节点的所谓"有限差方法". 中华先贤关于插值方法的研究比西方超前了上千年.

1.3 逐步插值过程

现在将 Lagrange 插值方法(算法 1.1)加工成某种简单计算过程的重复.

首先引进一套仅限于本节专用的数学符号: 对于给定的插值点 x, 记 $y_{i\cdots k}$ 表示依数据表 $(x_j, y_j), j=i, i+1, \cdots, k$ 生成的插值结果.

1.3.1 两点插值的松弛公式

设两对数据 (x_0, y_0), (x_1, y_1) 的插值结果为 y_{01}, 即有数据表(如表 1.5 所示)

表 1.5

x	x_0	x_1
y_{01}	y_0	y_1

则依式(4)有

$$y_{01}=\frac{x-x_1}{x_0-x_1}y_0+\frac{x-x_0}{x_1-x_0}y_1 \tag{9}$$

注意到数值 y_{01} 其实是 y_0，y_1 的加权平均，引进松弛因子

$$\omega_{01}=\frac{x-x_1}{x_0-x_1}$$

则有

$$\begin{aligned} y_{01} &= \omega_{01}y_0+(1-\omega_{01})y_1 \\ &= y_1+\omega_{01}(y_0-y_1) \end{aligned}$$

这说明，插值结果 y_{01} 可以看作是数据 y_1 适当修正的结果，其校正量等于偏差 y_0-y_1 的 ω_{01} 倍.

类似地，记 y_{12} 表示按表 1.6 所示数据表生成的插值结果.

表 1.6

x	x_1	x_2
y_{12}	y_1	y_2

则有

$$\begin{aligned} y_{12} &= \frac{x-x_2}{x_1-x_2}y_1+\frac{x-x_1}{x_2-x_1}y_2 \\ &= y_2+\omega_{12}(y_1-y_2) \end{aligned} \tag{10}$$

式中

$$\omega_{12}=\frac{x-x_2}{x_1-x_2}$$

1.3.2 三点插值的松弛过程

进一步考察三点插值. 记三组数据(x_0,y_0)，(x_1,y_1)，(x_2,y_2)的插值结果为 y_{02}(如表 1.7 所示)：

表 1.7

x	x_0	x_1	x_2
y_{02}	y_0	y_1	y_2

则依式(7)有

$$y_{02}=\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}y_0+\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}y_1$$
$$+\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}y_2 \tag{11}$$

自然会问,能否用两点插值的结果 y_{01},y_{12} 松弛生成三点插值的结果 y_{02} 呢?也就是说,是否存在某个松弛因子 ω_{02},使成立

$$\begin{aligned}y_{02}&=\omega_{02}y_{01}+(1-\omega_{02})y_{12}\\&=y_{12}+\omega_{02}(y_{01}-y_{12})\end{aligned} \tag{12}$$

依据式(9)、式(10)和式(11)不难发现,上式左端与右端均为 y_0,y_1,y_2 的线性组合(在插值点 x 给定的情况下),比较左右两端的组合系数容易定出松弛因子

$$\omega_{02}=\frac{x-x_2}{x_0-x_2}$$

代入式(12)知

$$y_{02}=\frac{x-x_2}{x_0-x_2}y_{01}+\frac{x-x_0}{x_2-x_0}y_{12} \tag{13}$$

由此可见,式(11)三点插值的插值结果 y_{02} 依式(13)亦可理解为两点插值的插值结果,也就是说,三点插值的数据表 1.7 等价于两点插值的数据表(如表 1.8 所示)

表 1.8

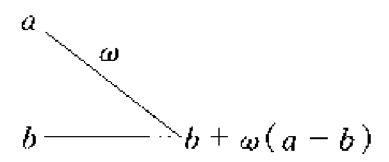
x	x_0	x_2
y_{02}	y_{01}	y_{12}

其中 y_{01},y_{12} 和 y_{02} 均为两点插值的计算结果:

$$\begin{aligned}y_{01}&=y_1+\omega_{01}(y_0-y_1),\quad \omega_{01}=\frac{x-x_1}{x_0-x_1}\\y_{12}&=y_2+\omega_{12}(y_1-y_2),\quad \omega_{12}=\frac{x-x_2}{x_1-x_2}\end{aligned} \tag{14}$$

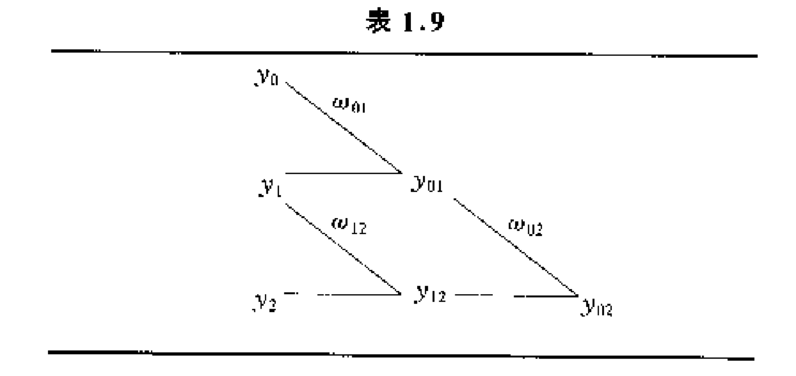
$$y_{02} = y_{12} + \omega_{02}(y_{01} - y_{12}), \quad \omega_{02} = \frac{x - x_2}{x_0 - x_2}$$

设引进计算格式



则式(14)的逐步插值过程可表述为**逐步插值表**,如表 1.9 所示.

表 1.9



这里又是"简单的重复生成复杂".

例 3 例 1 依据数据表 1.3 用两对数据 $x_1 = 3, y_1 = 1.732\ 05, x_2 = 4, y_2 = 2.000\ 00$ 插出 $x = 3.45$ 的结果 $y_{12} = 1.852\ 628$.若是改用两对数据 $x_0 = 2, y_0 = 1.414\ 21, x_1 = 3, y_1 = 1.732\ 05$ 则又可插出结果 $y_{01} = 1.875\ 078$,如果利用这两个结果按式(13)进行两点插值,得 $y_{02} = 1.858\ 80$.这一结果同例 2 三点插值的计算结果是一致的.

前已看到,三点插值可以化归为两点插值的重复,即通过两点插值得到结果.一般地,反复地计算两点插值即可生成多点 Lagrange 插值(8)的值.这就是所谓**逐步插值算法**.

下面再就 4 个节点的具体情形进一步列出逐步插值的计算公式.

1.3.3 多点插值的逐步松弛

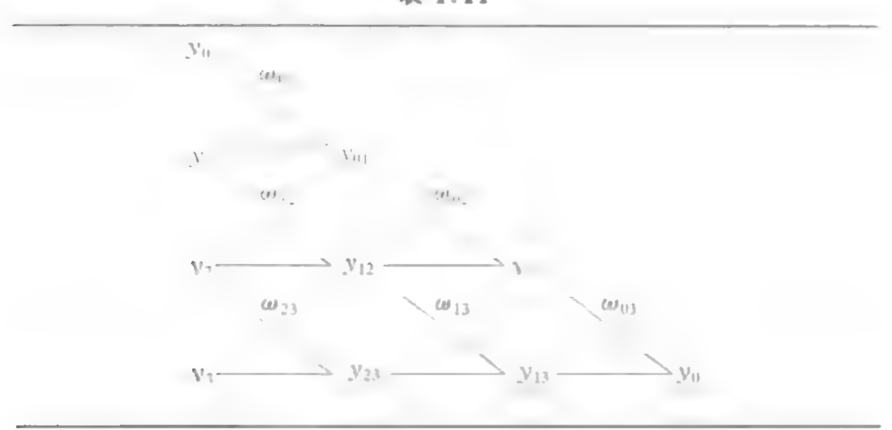
进一步考察 4 点插值(如表 1.10 所示)

表 1.10

x	x_0	x_1	x_2	x_3
y_{03}	y_0	y_1	y_2	y_3

不难证明,其插值结果 y_{03} 可以通过表 1.11 所示松弛过程逐步生成:

表 1.11



直接运用 Lagrange 公式(8)容易验证，对于表 1.11 中新增加的数据(用波纹线标出)，成立

$$
\begin{aligned}
y_{23} &= y_3 + \omega_{23}(y_2 - y_3),\ \omega_{23} = \frac{x - x_3}{x_2 - x_3} \\
y_{13} &= y_{23} + \omega_{13}(y_{12} - y_{23}),\ \omega_{13} = \frac{x - x_3}{x_1 - x_3} \\
y_{03} &= y_{13} + \omega_{03}(y_{02} - y_{13}),\ \omega_{03} = \frac{x - x_3}{x_0 - x_3}
\end{aligned}
\tag{15}
$$

以上加工手续可以推广到任意多个节点的一般情形. 总之，反复执行逐步松弛的加工手续，可以将多个节点的插值计算化归为两点插值过程的重复. 这里又一次体现了“简单的重复生成复杂”这一基本原理.

读者可以联系多项式求值的秦九韶算法(参看引论 0.2.3 小节)考虑逐步插值算法的理论背景. 为此再从函数逼近的角度考察插值方法.

1.4　插值逼近

1.4.1　插值逼近的概念

实际问题当中碰到的函数 $f(x)$ 可能很复杂，直接研究 $f(x)$ 往往很困难. 面对这种情况，一种很自然的想法是，设法将所考察的函数“简单化”，也就是说，构造某个简单函数 $g(x)$ 作为 $f(x)$ 的近似，然后处理 $g(x)$ 获得关于 $f(x)$ 所要的结果，这类方法称作**逼近方法**.

插值方法是逼近方法的一种. 如果要求逼近函数 $g(x)$ 与所逼近的复杂函数 $f(x)$ 在若干节点上取相同的离散信息——譬如取相同的函数值或导数值，这种逼近方法称作**插值方法**. 用插值方法逼近 $f(x)$ 的逼近函数 $g(x)$ 称作**插值**

函数.

可以选用不同类型的简单函数充当插值函数,如果限定插值函数为代数多项式,这类插值方法称为**代数插值**,相应的插值函数亦称**插值多项式**.

代数插值又分多种类型.

1.4.2 Taylor 插值

温故而知新.在介绍新的插值方法之前,首先回顾一下人们所熟知的 Taylor 展开方法.作为微积分方法核心内容的 Taylor 展开式其实就是一种插值公式.

众所周知,对于所考察的函数 $f(x)$,在给定点 x_0 邻近它可以用 Taylor 展开式 $p_n(x)$来逼近:

$$p_n(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n$$

这个多项式与 $f(x)$在点 x_0 具有相同的直到 n 阶导数值:

$$p_n^{(i)}(x_0) = f^{(i)}(x_0), \quad i = 0,1,\cdots,n \tag{16}$$

因而它可以看作下述 **Taylor 插值**的解:

问题 4 设已知 $f(x)$ 在点 x_0 处的导数值 $f^{(i)}(x_0)$,$i=0,1,\cdots,n$,求作 n 次式[①] $p_n(x)$使满足式(16).

例 4 作 $f(x)=\sqrt{x}$在 $x_0=4$ 的一次与二次 Taylor 多项式,并利用它们求 $x=\sqrt{3.45}$的近似值.

解 注意到

$$f(x)=\sqrt{x}, \; f'(x)=\frac{1}{2\sqrt{x}}, \; f''(x)=-\frac{1}{4x\sqrt{x}}$$

$$x_0=4, \; f(x_0)=2, \; f'(x_0)=0.25, \; f''(x_0)=-0.031\,25$$

作出一次、二次插值多项式

$$p_1(x)=2+0.25(x-4)$$

$$p_2(x)=2+0.25(x-4)-0.015\,625(x-4)^2$$

令 $x=3.45$ 分别求得近似值 $p_1(x)=1.862\,5$ 与 $p_2(x)=1.857\,77$,同准确值 $\sqrt{3.45}=1.857\,417\,56\cdots$比较,它们分别有 2 位和 4 位有效数字.

1.4.3 Lagrange 插值

前述 Taylor 插值要求插值函数 $p(x)$与所逼近的函数 $f(x)$在展开点 x_0 具有相同的直到 n 阶导数值,这项要求很苛刻,函数 $f(x)$必须相当简单才行.

---

① 本章所说的 n 次式,往往泛指次数$\leqslant n$ 的多项式,在特殊情况下其次数可能小于 n

为使插值方法便于使用,可增添函数值来替代所要提供的导数值.如果要求插值函数 $p(x)$与所逼近的函数 $f(x)$在一系列节点上取相同的函数值,这种插值方法称作 **Lagrange 插值**.

Lagrange 插值的提法是:

问题 5 设已知 $f(x)$的函数值 $f(x_i), i=0,1,\cdots,n$,求作 n 次式 $p_n(x)$使满足

$$p_n(x_i)=f(x_i), \quad i=0,1,\cdots,n \tag{17}$$

解 其实,这里所要构造的 n 次式 $p_n(x)$在 1.2 节早已给出.进一步考察 Lagrange 插值公式(8),如果视 x 为变量,则式(8)可理解为关于 x 的 n 次式,记作 $p_n(x)$,即

$$p_n(x) = \sum_{k=0}^{n}\lambda_k(x)f(x_k),$$

$$\lambda_k(x) = \prod_{\substack{j=0\\ j\neq k}}^{n}\frac{x-x_j}{x_k-x_j}, \quad k=0,1,\cdots,n$$

需要强调指出的是,其中的每个 $\lambda_k(x)$都是 n 次式,且满足条件

$$\lambda_k(x_k)=1$$

$$\lambda_k(x_j)=0, \quad j=0,1,\cdots,k-1,k+1,\cdots,n$$

通常称这些 $\lambda_k(x)$为 **Lagrange 插值基函数**.考虑到 $\lambda_k(x)$在节点上取特殊值,立即得知

$$p_n(x_i) = \sum_{k=0}^{n}\lambda_k(x_i)f(x_k) = f(x_i), \quad i=0,1,\cdots,n$$

因而 $p_n(x)$即为问题 5 的解.

1.4.4 Hermite 插值

前面考察了两种代数插值:Taylor 插值要求插值函数与原来的函数在某一点上有相同的导数值,而 Lagrange 插值则要求两者在多个节点上有相同的函数值.其实它们是两种极端的情况.

在某些实际问题中,为了保证插值函数能更好地密合原来的函数,不但要求"过点",即两者在节点上具有相同的函数值,而且要求"相切",即在节点上还具有相同的导数值.这类插值称作**切触插值**,或称 **Hermite 插值**.

显然,Hermite 插值是 Lagrange 插值与 Taylor 插值的综合与推广.这里考察两个简单的 Hermite 插值问题,以为下一节的样条插值做准备.

问题 6 求作二次式 $p_2(x)$使满足条件

$$\begin{aligned} &p_2(x_0)=y_0, \ p_2'(x_0)=y_0' \\ &p_2(x_1)=y_1 \end{aligned} \tag{18}$$

解 下面提供三条途径构造 $p_2(x)$.

1. **待定系数法**

设所求多项式为

$$p_2(x)=a_0x^2+a_1x+a_2$$

依插值条件(18)可列出方程组

$$\begin{cases}a_0x_0^2+a_1x_0+a_2=y_0\\ a_0x_1^2+a_1x_1+a_2=y_1\\ 2a_0x_0+a_1=y_0'\end{cases}$$

据此解出系数 a_0,a_1,a_2 即得所求的 $p_2(x)$.

构造插值多项式的待定系数法是一种代数化方法,这种方法有普适性,只是所归结出的代数方程组往往形式比较复杂.

2. **余项校正法**

再考察插值条件(18),注意到满足条件 $p_1(x_0)=y_0,p_1(x_1)=y_1$ 的插值多项式是

$$p_1(x)=\frac{x-x_1}{x_0-x_1}y_0+\frac{x-x_0}{x_1-x_0}y_1$$

设法用某个"余项"校正 $p_1(x)$以获得所求的 $p_2(x)$,设令

$$p_2(x)=p_1(x)+c(x-x_0)(x-x_1)$$

则不管余项系数 c 怎样取值,总有 $p_2(x_0)=y_0,p_2(x_1)=y_1$,再用剩下的一个条件 $p_2'(x_0)=y_0'$ 确定系数 c 即得所求的 $p_2(x)$.

运用余项校正法构造插值多项式,目的在于尽可能地减少待定系数的个数,从而使所归结出的代数方程组比较容易求解.

3. **基函数方法**

基函数方法的设计机理是,将插值多项式的构造化归为求解几个特殊数据表的插值问题.

为简化处理,先设 $x_0=0,x_1=1$ 而令所求的 $p_2(x)$具有形式

$$p_2(x)=y_0\varphi_0(x)+y_1\varphi_1(x)+y_0'\psi_0(x)$$

式中基函数 $\varphi_0(x),\varphi_1(x),\psi_0(x)$均为二次式,它们分别满足条件

$$\varphi_0(0)=1,\quad \varphi_0(1)=\varphi_0'(0)=0$$

$$\varphi_1(1)=1,\quad \varphi_1(0)=\varphi_1'(0)=0$$

$$\psi_0'(0)=1,\quad \psi_0(0)=\psi_0(1)=0$$

满足这些条件的插值多项式很容易构造出来.事实上,由条件 $\psi_0(0)=\psi_0(1)=0$ 知 $\psi_0(x)$有两个零点 $x=0,1$,因而它具有形式

$$\psi_0(x)=cx(x-1)$$

再利用剩下的一个条件 $\psi_0'(0)=1$ 定出 $c=-1$，于是有

$$\psi_0(x)=x(1-x)$$

又，由条件 $\varphi_0(1)=0$ 知 $\varphi_0(x)$ 有一个零点 $x=1$，故它具有形式

$$\varphi_0(x)=(ax+b)(x-1)$$

再用剩下的两个条件 $\varphi_0(0)=1,\varphi_0'(0)=0$ 可列出方程组

$$\begin{cases}-b=1\\-a+b=0\end{cases}$$

由此得知 $a=b=-1$，从而有

$$\varphi_0(x)=1-x^2$$

同理有

$$\varphi_1(x)=x^2$$

这就针对 $x_0=0,x_1=1$ 的特殊情形构造出所求的插值多项式.

一般地，如果 x_0,x_1 是随意给出的两个节点，则通过变换 $t=\dfrac{x-x_0}{h}$，$h=x_1-x_0$ 即可变到节点为 0,1 的情形，因而问题 6 所求的插值多项式具有形式

$$p_2(x)=y_0\varphi_0\left(\frac{x-x_0}{h}\right)+y_1\varphi_1\left(\frac{x-x_0}{h}\right)+hy_0'\psi_0\left(\frac{x-x_0}{h}\right)$$

基函数方法的设计思想依然是尽量简化所归结出的代数方程组. 如果所考察的插值问题具有对称结构，则往往首选基函数方法. 前述 Lagrange 插值的问题 5 是这种情形. 下面再举一个 Hermite 插值的例子.

问题 7　求作三次式 $p_3(x)$ 使满足条件

$$p_3(x_0)=y_0,\quad p_3(x_1)=y_1$$
$$p_3'(x_0)=y_0',\quad p_3'(x_1)=y_1'$$

解　这个问题有对称结构，考虑用基函数方法求解. 记 $h=x_1-x_0$，令

$$p_3(x)=y_0\varphi_0\left(\frac{x-x_0}{h}\right)+y_1\varphi_1\left(\frac{x-x_0}{h}\right)+hy_0'\psi_0\left(\frac{x-x_0}{h}\right)+hy_1'\psi_1\left(\frac{x-x_0}{h}\right)\quad(19)$$

式中 $\varphi_0(x),\varphi_1(x),\psi_0(x),\psi_1(x)$ 是插值基函数，它们分别满足插值条件

$$\varphi_0(0)=1,\varphi_0(1)=\varphi_0'(0)=\varphi_0'(1)=0$$
$$\varphi_1(1)=1,\varphi_1(0)=\varphi_1'(0)=\varphi_1'(1)=0$$
$$\psi_0'(0)=1,\psi_0(0)=\psi_0(1)=\psi_0'(1)=0$$
$$\psi_1'(1)=1,\psi_1(0)=\psi_1(1)=\psi_1'(0)=0$$

作为习题，请读者自行导出这些插值基函数，结果是

$$\varphi_0(x)=(x-1)^2(2x+1),\quad \varphi_1(x)=x^2(-2x+3)$$

$$\psi_0(x)=x(x-1)^2,\quad \psi_1(x)=x^2(x-1)$$

1.5 样条插值

从20世纪60年代初开始，首先由于航空、造船等工程设计的需要，发展了所谓样条函数方法. 今天，这种方法已成为数值逼近的一个极其重要的分支. 在外形设计乃至计算机辅助设计的许多领域，样条函数都被认为是一种有效的数学工具.

1.5.1 样条函数的概念

样条函数对于人们并不陌生，常用的阶梯函数（图1.2）和折线函数（图1.3）分别是简单的零次样条和一次样条.

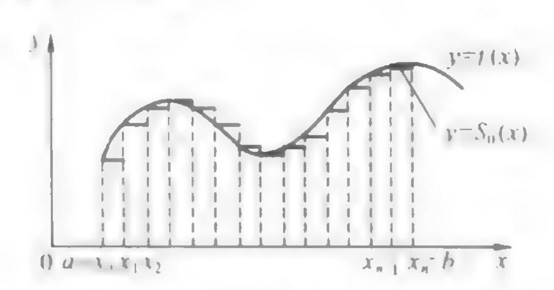


图 1.2 作为零次样条的阶梯函数

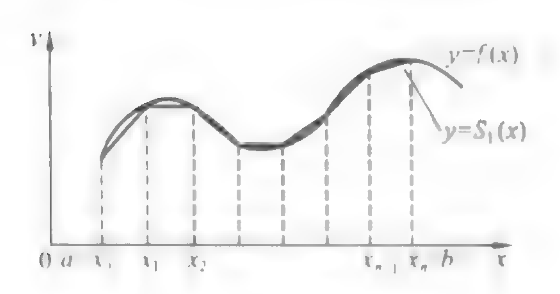


图 1.3 作为一次样条的折线函数

不难抽象出零次样条（阶梯函数）和一次样条（折线函数）的数学定义. 对于区间$[a,b]$的某个**分划**

$$\Delta:\ a=x_0<x_1<\cdots<x_n=b$$

定义 2 称 $S_0(x)$ 为具有分划 Δ 的**零次样条**，如果它在分划 Δ 的每个子段 $[x_i,x_{i+1}]$ $(i=0,1,\cdots,n-1)$ 上都是零次式（即取定值）；而称 $S_1(x)$ 为具有分划 Δ 的**一次样条**，如果它在每个子段 $[x_i,x_{i+1}]$ 上都是一次式，且在每个内结点

$x_i(i=1,2,\cdots,n-1)$函数值连续，即成立

$$S_1(x_i-0)=S_1(x_i+0),\quad i=1,2,\cdots,n-1$$

所谓样条函数，从数学上讲，就是按一定的光滑性要求"装配"起来的分段多项式。需要注意的是，光滑性的要求不能"过分"。譬如，如果进一步要求一次样条$S_1(x)$在每个内结点都具有连续的一阶导数，则它便退化为区间$[a,b]$上的一次式。

定义 3 称$S_2(x)$为具有分划Δ的**二次样条**，如果它在每个子段$[x_i,x_{i+1}]$上都是二次式，且在内结点连续且具有连续的一阶导数，即成立

$$\begin{aligned}S_2(x_i-0)&=S_2(x_i+0),\\ S_2'(x_i-0)&=S_2'(x_i+0),\end{aligned}\quad i=1,2,\cdots,n-1$$

定义 4 称$S_3(x)$为具有分划Δ的**三次样条**，如果它在每个子段$[x_i,x_{i+1}]$上都是三次式，且在内结点上具有直到二阶连续导数

$$\begin{aligned}S_3(x_i-0)&=S_3(x_i+0),\\ S_3'(x_i-0)&=S_3'(x_i+0),\quad i=1,2,\cdots,n-1\\ S_3''(x_i-0)&=S_3''(x_i+0),\end{aligned}$$

样条函数的特点是，它既是充分光滑的，同时又保留有一定的间断性。光滑性保证了外形曲线的平滑优美，而间断性则使它能转折自如地被灵活运用。

样条函数概念来源于工程设计的实践。所谓"样条"(Spline)是工程设计中的一种绘图工具，它是富有弹性的细长条。绘图时，绘图员用压铁迫使样条通过指定的型值点(x_i,y_i)，并且调整样条使它具有光滑的外形。这种外形曲线可以看作是作为弹性细梁的样条，在压铁的集中载荷作用下产生的挠度曲线。在挠度不大的情况下，它恰好表示为上述定义的三次样条函数，压铁的作用点就是样条函数的结点。

1.5.2 三次样条插值

样条插值其实是一种改进的分段插值。特别地，由于折线函数就是一次样条，因此就一次插值而言，样条插值和分段插值是一回事。

下面将主要研究三次样条插值。为了正确地提出问题，首先分析三次样条所具有的自由度。

对于具有分划$\Delta: a=x_0<x_1<\cdots<x_n=b$的三次样条函数$S_3(x)$，由于它在每个子段上都是三次式，总计有$4n$个待定参数，但为要保证在每个结点处连续且有连续的一阶和二阶导数，必须附加$3(n-1)$个光滑性约束条件，因而$S_3(x)$的自由度为

$$4n-3(n-1)=n+3$$

就是说,为了具体确定具有分划Δ的三次样条函数,必须再补充给出 $n+3$ 个条件.

下面具体考察三次样条插值.

问题 8 求作具有分划Δ的三次样条 $S_3(x)$,使满足

$$S_3(x_i)=y_i,\quad i=0,1,\cdots,n \tag{20}$$

$$S_3'(x_0)=y_0',\quad S_3'(x_n)=y_n' \tag{21}$$

解 样条函数的构造用待定系数法.问题在于参数的选择.由于 $S_3(x)$在每个子段上都是三次式,而每个三次式有 4 个系数,这样共需要确定 $4n$ 个系数.因此,虽然原则上可选取分段多项式的系数作为待定参数,但这种方法所归结出的是大规模的稠密方程组.

为简化计算,这里选取结点上的导数值 $S_3'(x_i)=m_i$ 作为参数,按上一节的式(19)

$$S_3(x)=\varphi_0\left(\frac{x-x_i}{h_i}\right)y_i+\varphi_1\left(\frac{x-x_i}{h_i}\right)y_{i+1}$$

$$+h_i\psi_0\left(\frac{x-x_i}{h_i}\right)m_i+h_i\psi_1\left(\frac{x-x_i}{h_i}\right)m_{i+1},\quad x_i<x<x_{i+1} \tag{22}$$

式中 $h_i=x_{i+1}-x_i$,而

$$\varphi_0(x)=(x-1)^2(2x+1),\quad \varphi_1(x)=x^2(-2x+3),$$

$$\psi_0(x)=x(x-1)^2,\quad \psi_1(x)=x^2(x-1)$$

不管参数 m_i 怎样取值,这样构造出的 $S_3(x)$在每个结点 $x_i(1\leqslant i\leqslant n-1)$上必定连续且有连续的一阶导数.现在的问题是,怎样选取参数 m_i 的值使其二阶导数也连续.

对式(22)求导两次,易得

$$S_3''(x)=\frac{6}{h_i^2}\left[2\left(\frac{x-x_i}{h_i}\right)-1\right]y_i-\frac{6}{h_i^2}\left[2\left(\frac{x-x_i}{h_i}\right)-1\right]y_{i+1}$$

$$+\frac{1}{h_i}\left[6\left(\frac{x-x_i}{h_i}\right)-4\right]m_i+\frac{1}{h_i}\left[6\left(\frac{x-x_i}{h_i}\right)-2\right]m_{i+1}$$

因之,在子段$[x_i,x_{i+1}]$的左右两端分别有

$$S_3''(x_i)=6\frac{y_{i+1}-y_i}{h_i^2}-\frac{4m_i+2m_{i+1}}{h_i} \tag{23}$$

$$S_3''(x_{i+1})=-6\frac{y_{i+1}-y_i}{h_i^2}+\frac{2m_i+4m_{i+1}}{h_i} \tag{24}$$

为了保证二阶导数的连续性

$$S_3''(x_i-0)=S_3''(x_i+0),\quad i=1,2,\cdots,n-1$$

式(23)与式(24)应当相容,因而应有

$$\frac{m_{i-1}+2m_i}{h_{i-1}}+\frac{2m_i+m_{i+1}}{h_i}=3\left(\frac{y_i-y_{i-1}}{h_{i-1}^2}+\frac{y_{i+1}-y_i}{h_i^2}\right) \tag{25}$$

令

$$\alpha_i=\frac{h_{i-1}}{h_{i-1}+h_i},\quad \beta_i=3\left[(1-\alpha_i)\frac{y_i-y_{i-1}}{h_{i-1}}+\alpha_i\frac{y_{i+1}-y_i}{h_i}\right]$$

则式(25)可表为

$$(1-\alpha_i)m_{i-1}+2m_i+\alpha_i m_{i+1}=\beta_i \tag{26}$$

另外,由条件(21)直接给出

$$m_0=y_0',\quad m_n=y_n'$$

据此从式(26)中消去 m_0 和 m_n 即可归结出关于参数 $m_1,m_2,\cdots,m_{n-1}$ 的方程组

$$\begin{cases}2m_1+\alpha_1 m_2=\beta_1-(1-\alpha_1)y_0'\\(1-\alpha_i)m_{i-1}+2m_i+\alpha_i m_{i+1}=\beta_i, & i=2,3,\cdots,n-2\\(1-\alpha_{n-1})m_{n-2}+2m_{n-1}=\beta_{n-1}-\alpha_{n-1}y_n'\end{cases} \tag{27}$$

这种形式的方程组称作样条插值的**基本方程组**,这类方程组由于其系数矩阵

$$A=\begin{bmatrix}2 & \alpha_1 & & & \mathbf{0}\\1-\alpha_2 & 2 & \alpha_2 & & \\ & \ddots & \ddots & \ddots & \\ & & 1-\alpha_{n-2} & 2 & \alpha_{n-2}\\ \mathbf{0} & & & \alpha_{n-1} & 2\end{bmatrix}$$

的非零元素集中在三条对角线上而被称作是**三对角型**的.求解这类方程组的一种有效方法是所谓追赶法(参看第六章6.1节).

综上所述,样条插值的计算过程分两步:先求解基本方程组(27)确定参数 m_i,然后利用显式公式(22)求出插值结果.

算法1.2(样条插值)

设给定数据表

表1.12

x	x_0	x_1	x_2	$\cdots$	x_n
y	y_0	y_1	y_2	$\cdots$	y_n
y'	y_0'				y_n'

求解方程组(27)确定参数 $m_i(i=1,2,\cdots,n-1)$,然后依式(22)计算插值结果 $y=S_3(x)$.

例 5　对于函数 $f(x)=\dfrac{1}{1+x^2}$,取等距节点 $x_i=-5+i, i=0,1,\cdots,10$,设已给出节点上的函数值以及左右两个端点的一阶导数值,按上述样条函数方法进行插值.计算结果见表 1.13.

表 1.13

x	$f(x)$	$S_3(x)$	x	$f(x)$	$S_3(x)$
-5.0	0.038 46	0.038 46	-2.3	0.158 98	0.241 45
4.8	0.041 60	0.037 58	-2.0	0.200 00	0.200 00
-4.5	0.047 60	0.042 48	-1.8	0.235 85	0.188 78
-4.3	0.051 31	0.048 42	-1.5	0.307 69	0.235 35
4.0	0.058 82	0.058 82	-1.3	0.371 75	0.316 50
3.8	0.064 77	0.065 56	-1.0	0.500 00	0.500 00
-3.5	0.075 47	0.076 06	-0.8	0.609 76	0.643 16
-3.3	0.084 10	0.084 26	0.5	0.800 00	0.843 40
3.0	0.100 00	0.100 00	-0.3	0.917 43	0.940 90
-2.8	0.113 12	0.113 66	0	1.000 00	1.000 00
2.5	0.137 93	0.139 71			

小　　结

本章从两个不同的角度考察插值问题.

所谓"插值",通俗地说,就是在所给数据表中再插进一些所需要的函数值.其实,中国古代数学家早就理解并掌握了数据加工的插值原理.为要将所给数据表加工成所求的插值结果 y,可以直接套用显式的计算公式(如 Lagrange 公式),也可以反复施行两点插值逐步地递推计算.

微积分问世以后,插值方法又被理解为一种逼近函数的构造方法.这种方法构造出的插值函数与所逼近的复杂函数取某些相同的离散数据,譬如在某些节点上取相同的函数值或导数值.插值函数可以是普通的代数多项式,这类插值称代数插值.Hermite 插值及其特例 Lagrange 插值与 Taylor 插值都是代数插值.而样条插值则取分段多项式作为插值函数.

针对难以处理的复杂函数 $f(x)$,构造出它的插值函数 $g(x)$ 以后,可以处理 $g(x)$ 获得 $f(x)$ 的有关信息.譬如计算 $g'(x)$ 近似 $f'(x)$ 等等.这就简化了处理手续.

本章处理插值问题基于笛卡儿(Descarles)的代数化方法.

笛卡儿认为,最有价值的知识是科学方法.笛卡儿倡导的科学方法其要点之

一是数学的代数化.他发明了直角坐标系,沟通了几何曲线与代数方程,统一了几何与代数这两大学科,从而为数学的发展开辟了广阔的前景.

本书关于数值微积分的讨论正是在数学问题代数化的背景下展开的.

本章研究插值方法的基本策略是,基于代数精度的概念,先将所要考察的插值问题化归为确定某些参数的代数问题,进而依据所给的插值条件列出代数方程,解之即得所求的插值公式.

为简化处理手续,本章突出地强调了基函数方法.基函数方法本质上是推广了的坐标系方法.这种方法将一般形式的插值问题化归为某些特定条件的插值问题,后者比较容易求解.

题解 1.1 Lagrange 插值基函数

提要 对于给定的一组节点 $x_i(i=0,1,\cdots,n)$,Lagrange 插值基函数 $l_i(x)$ $(i=0,1,\cdots,n)$ 是这样一组 n 次式,它们在所给节点上取特殊值

$$l_i(x_i)=1$$
$$l_i(x_j)=0,\quad 当\ j\neq i\ 时$$

容易看出,$l_i(x)$ 有显式表达式(这里 $\prod\limits_{j\neq i}$ 表示 $\prod\limits_{\substack{j=0\\ j\neq i}}^{n}$)

$$l_i(x)=\prod_{j\neq i}\frac{x-x_j}{x_i-x_j},\quad i=0,1,\cdots,n$$

题 1 证明对于任给互异节点 $x_i(i=0,1,\cdots,n)$ 下列恒等式成立:

$$\sum_{i=0}^{n}\left(\prod_{j\neq i}\frac{x-x_j}{x_i-x_j}\right)x_i^k\equiv x^k,\quad k=0,1,\cdots,n$$

证 据 Lagrange 插值多项式的唯一性知,当 $k\leqslant n$ 时幂函数 $f(x)=x^k$ 关于 $n+1$ 个节点 $x_i(i=0,1,\cdots,n)$ 的插值多项式就是它自身,故依 Lagrange 公式有

$$\sum_{i=0}^{n}x_i^k l_i(x)=\sum_{i=0}^{n}\left(\prod_{j\neq i}\frac{x-x_j}{x_i-x_j}\right)x_i^k\equiv x^k,\quad k=0,1,\cdots,n$$

特别地,当 $k=0$ 时有

$$\sum_{i=0}^{n}l_i(x)=\sum_{i=0}^{n}\prod_{j\neq i}\frac{x-x_j}{x_i-x_j}\equiv 1$$

而当 $k=1$ 时则有

$$\sum_{i=0}^{n}x_i l_i(x)=\sum_{i=0}^{n}\left(\prod_{j\neq i}\frac{x-x_j}{x_i-x_j}\right)x_i\equiv x$$

题 2 证明下列恒等式成立:

(1) $\sum_{i=0}^{n}\prod_{j\neq i}\frac{x-j}{i-j}\equiv 1$

(2) $\sum_{i=0}^{n}\prod_{j\neq i}\frac{x-j}{i-j}\cdot i\equiv x$

证 上题令 $x_i=i(i=0,1,\cdots,n)$即得.

题 3 设 $l_i(x)(i=0,1,\cdots,n)$是以 $x_i(i=0,1,\cdots,n)$为节点的 Lagrange 插值基函数,证明

$$\sum_{i=0}^{n}(x_i-x)^k l_i(x)\equiv 0,\quad k=1,2,\cdots,n$$

证 将$(x_i-x)^k$二项式展开,据题 1 有

$$\begin{aligned}\sum_{i=0}^{n}(x_i-x)^k l_i(x)&=\sum_{i=0}^{n}\left[\sum_{j=0}^{k}\binom{k}{j}x_i^j(-x)^{k-j}\right]l_i(x)\\&=\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}x^{k-j}\left[\sum_{i=0}^{n}x_i^j l_i(x)\right]\\&=\sum_{j=0}^{k}(-1)^{k-j}\binom{k}{j}x^{k-j}x^j\\&=x^k\sum_{j=0}^{k}\binom{k}{j}(-1)^{k-j}=x^k(1-1)^k=0\end{aligned}$$

题 4 证明当 $m>n$ 时成立

$$\sum_{i=0}^{n}(-1)^{n-i}\frac{C_m^n C_n^i}{m-i}=\frac{1}{m-n}$$

其中

$$C_m^n=\frac{m!}{n!\,(m-n)!},\quad C_n^i=\frac{n!}{i!\,(n-i)!}$$

证 据题 1 知

$$\sum_{i=0}^{n}l_i(x)=\sum_{i=0}^{n}\prod_{j\neq i}\frac{x-x_j}{x_i-x_j}\equiv 1$$

令 $x_j=j$ 且取 $x=m$ 则有

$$\begin{aligned}l_i(m)&=\prod_{\substack{j=0\\j\neq i}}^{n}\frac{m-j}{i-j}\\&=\frac{m(m-1)\cdots(m-i+1)(m-i-1)\cdots(m-n)}{i\times(i-1)\times\cdots\times 1\times(-1)\times\cdots\times(i-n)}\\&=\frac{m!\,(m-n)}{(m-n)!\,(m-i)}\cdot\frac{1}{i!\,(-1)^{n-i}(n-i)!}\end{aligned}$$

于是有

$$1=\sum_{i=0}^{n}l_i(m)=(m-n)\sum_{i=0}^{n}(-1)^{n-i}\frac{C_m^n C_n^i}{m-i}$$

题 5 对于给定的二元函数 $f(x,y)$，求作二元一次式 $u(x,y)$，使在给定点 $(x_i,y_i)(i=0,1,2)$ 与 $f(x,y)$ 取相同的函数值，即满足插值条件

$$u(x_i,y_i)=f(x_i,y_i),\quad i=0,1,2$$

解 用基函数方法. 首先构造二元一次式 $l_0(x,y)$，使满足

$$l_0(x_0,y_0)=1,\quad l_0(x_1,y_1)=l_0(x_2,y_2)=0$$

满足这些特定条件的 $l_0(x,y)$ 很容易构造出来，结果是

$$l_0(x,y)=\begin{vmatrix}1 & x & y\\ 1 & x_1 & y_1\\ 1 & x_2 & y_2\end{vmatrix}\Bigg/\begin{vmatrix}1 & x_0 & y_0\\ 1 & x_1 & y_1\\ 1 & x_2 & y_2\end{vmatrix}$$

考虑到节点地位对等，此外有

$$l_1(x,y)=\begin{vmatrix}1 & x_0 & y_0\\ 1 & x & y\\ 1 & x_2 & y_2\end{vmatrix}\Bigg/\begin{vmatrix}1 & x_0 & y_0\\ 1 & x_1 & y_1\\ 1 & x_2 & y_2\end{vmatrix}$$

$$l_2(x,y)=\begin{vmatrix}1 & x_0 & y_0\\ 1 & x_1 & y_1\\ 1 & x & y\end{vmatrix}\Bigg/\begin{vmatrix}1 & x_0 & y_0\\ 1 & x_1 & y_1\\ 1 & x_2 & y_2\end{vmatrix}$$

用给定数据 $f(x_i,y_i)$ 将这些 Lagrange 插值基函数组合在一起，即得所求的插值多项式 $u(x,y)$：

$$u(x,y)=\sum_{i=0}^{2}f(x_i,y_i)l_i(x,y)$$

题解 1.2 插值多项式的构造

提要 如前文 1.4 节所指出的，插值多项式的构造方法有待定系数法、余项校正法和基函数方法三种. 具体情况具体分析，解题时应针对问题的特点选取合适的方法.

题 1 试构造次数≤3 的多项式 $p(x)$，使满足插值条件

$$p(0)=0,\quad p'(0)=1$$
$$p(1)=1,\quad p'(1)=2$$

解 这个插值问题很简单，考虑用待定系数法求解，令所求插值多项式

$$p(x)=a_0+a_1x+a_2x^2+a_3x^3$$
$$p'(x)=a_1+2a_2x+3a_3x^2$$

依所给插值条件有

$$0 = p(0) = a_0$$
$$1 = p'(0) = a_1$$
$$1 = p(1) = a_0 + a_1 + a_2 + a_3$$
$$2 = p'(1) = a_1 + 2a_2 + 3a_3$$

由此解出

$$a_0 = 0,\ a_1 = 1,\ a_2 = -1,\ a_3 = 1$$

故所求的插值多项式

$$p(x) = x - x^2 + x^3$$

题 2 设 $x_1 \neq \frac{x_0 + x_2}{2}$,求作次数≤2 的多项式 $p(x)$,使满足插值条件

$$p(x_0) = y_0,\ p'(x_1) = y_1',\ p(x_2) = y_2$$

解 注意到满足条件

$$q(x_0) = y_0,\ q(x_2) = y_2$$

的插值多项式

$$q(x) = y_0 + \frac{y_2 - y_0}{x_2 - x_0}(x - x_0)$$

试用余项校正法校正 $q(x)$得出所求的 $p(x)$,为此令

$$p(x) = q(x) + c(x - x_0)(x - x_2)$$

注意到

$$p'(x) = \frac{y_2 - y_0}{x_2 - x_0} + c(2x - x_0 - x_2)$$

依插值条件 $p'(x_1) = y_1'$ 可列出方程

$$\frac{y_2 - y_0}{x_2 - x_0} + c(2x_1 - x_0 - x_2) = y_1'$$

据此定出余项系数

$$c = \frac{y_1' - \dfrac{y_2 - y_0}{x_2 - x_0}}{2x_1 - x_0 - x_2}$$

这里要求 $2x_1 - x_0 - x_2 \neq 0$,即

$$x_1 \neq \frac{x_0 + x_2}{2}$$

题 3 求作次数≤5 的多项式 $p(x)$,使满足插值条件

$$p(0) = p(1) = p(2) = p(3) = p(4) = p'(0) = 1$$

解 依所给插值条件自然令

$$p(x) = 1 + cx(x-1)(x-2)(x-3)(x-4)$$

再利用条件 $p'(0)=1$ 可定出 $c=\frac{1}{24}$.

题 4　求作次数≤5 的多项式 $p(x)$,使满足插值条件

x_i	0	1	2
y_i	2	1	2
y_i'	-2	-1	
y_i''	10		

解　注意到满足插值条件

$$q(0)=2,\ q'(0)=-2,\ q''(0)=-10$$

的 Taylor 多项式

$$q(x)=-5x^2-2x+2$$

令

$$p(x)=-5x^2-2x+2+x^3(ax^2+bx+c)$$

由于

$$p'(x)=-10x-2+3x^2(ax^2+bx+c)+x^3(2ax+b)$$

用剩下的插值条件列出方程

$$1=p(1)=-5+(a+b+c)$$
$$-1=p'(1)=-12+3(a+b+c)+(2a+b)$$
$$2=p(2)=-22+8(4a+2b+c)$$

解之有

$$a=4,\ b=-15,\ c=17$$

于是所求的插值多项式

$$p(x)=4x^5-15x^4+17x^3-5x^2-2x+2$$

题 5　求作首项系数为 1 的 4 次式 $p(x)$,使满足条件

$$p(a)=p'(a)=p''(a)=0$$
$$p'(b)=0$$

解　令所求的 $p(x)$具有形式

$$p(x)=(x-a)^3(x-c)$$

由于

$$p'(x)=3(x-a)^2(x-c)+(x-a)^3$$

据条件 $p'(b)=0$ 定出

$$c=\frac{4b-a}{3}$$

故有

$$p(x)=(x-a)^3\left(x-\frac{4b-a}{3}\right)$$

题 6 求作前文 1.4 节问题 7 的插值基函数 $\varphi_0(x)$ 与 $\varphi_1(x)$，它们是三次式，分别满足插值条件

$$\varphi_0(0)=1,\ \varphi_0(1)=\varphi_0'(0)=\varphi_0'(1)=0$$

$$\varphi_1(1)=1,\ \varphi_1(0)=\varphi_1'(0)=\varphi_1'(1)=0$$

解 由条件 $\varphi_0(1)=\varphi_0'(1)=0$ 知 $\varphi_0(x)$ 具有形式

$$\varphi_0(x)=(x-1)^2(ax+b)$$

注意到

$$\varphi_0'(x)=2(x-1)(ax+b)+a(x-1)^2$$

用剩下的条件 $\varphi_0(0)=1,\varphi_0'(0)=0$ 可列出方程组

$$\begin{cases} b=1 \\ -2b+a=0 \end{cases}$$

由此定出 $a=2,b=1$，从而有

$$\varphi_0(x)=(x-1)^2(2x+1)$$

类似地知

$$\varphi_1(x)=x^2(-2x+3)$$

习 题 一

1. 证明：如果插值公式

$$f(x)\approx\sum_{i=0}^{n}\lambda_i f(x_i)$$

对幂函数 $f(x)=x^k,k=0,1,\cdots,m$ 准确成立，则它必对任给次数 $\leqslant m$ 的多项式准确成立.

2. 记

$$V(x_0,x_1,\cdots,x_{n-1},x)=\begin{vmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^n \\ 1 & x & x^2 & \cdots & x^n \end{vmatrix}$$

证明下列关系式成立：

(1) $V(x_0,x_1,\cdots,x_{n-1},x)=V(x_0,x_1,\cdots,x_{n-1})\prod_{j=0}^{n-1}(x-x_j)$

(2) $V(x_0,x_1,\cdots,x_n) = V(x_0,x_1,\cdots,x_{n-1})\prod_{j=0}^{n-1}(x_n - x_j)$

(3) $V(x_0,x_1,\cdots,x_n) = \prod_{j<i}(x_i - x_j)$

(4) $V(1,2,\cdots,n) = 1!2!\cdots(n-1)!$

3. 试针对两点插值问题画出 Lagrange 插值基函数的图形.

4. 设 $x_0 \neq x_1$，求作偶函数的二次式 $p(x)$，使满足条件

$$p(x_0) = f(x_0)\ ,\ p(x_1) = f(x_1)$$

5. 依据下列数据表所构造出的插值多项式 $p(x)$ 有多少次？为什么？请具体给出 $p(x)$ 的表达式：

(1)

x_i	-2	1	0	1	2	3
y_i	5	1	1	1	7	25

(2)

x_i	0	$\frac{1}{2}$	1	$\frac{3}{2}$	2	$\frac{5}{2}$
y_i	-1	$\frac{3}{4}$	0	$\frac{5}{4}$	3	$\frac{21}{4}$

6. 求作次数 $\leqslant 2$ 的多项式 $p(x)$，使满足条件

$$p(0)=1,\ p(1)=2,\ p'(0)=0$$

7. 求作次数 $\leqslant 3$ 的多项式 $p(x)$，使满足条件

$$p(x_i) = f(x_i),\quad i=0,1,2$$
$$p'(x_1) = f'(x_1)$$

8. 求作次数 $\leqslant 3$ 的多项式 $p(x)$，使满足条件

$$p(x_0) = f(x_0),\ p'(x_0) = f'(x_0)$$
$$p''(x_0) = f''(x_0),\ p(x_1) = f(x_1)$$

9. 求作次数 $\leqslant 4$ 的多项式 $p(x)$，使满足条件

$$p(0) = -1,\ p'(0) = -2$$
$$p(1) = 0,\ p'(1) = 10,\ p''(1) = 40$$

10. 求作前文 1.4 节问题 7 的插值基函数 $\psi_0(x),\psi_1(x)$，它们是三次式，分别满足条件

$$\psi_0'(0) = 1,\ \psi_0(0) = \psi_0(1) = \psi_0'(1) = 0$$
$$\psi_1'(1) = 1,\ \psi_1(0) = \psi_1(1) = \psi_1'(0) = 0$$

11. 设给定分划点 $-1,0,1$，试用待定系数法构造满足下列条件的三次样条 $S_3(x)$：

$$S_3(-1) = y_{-1},\ S_3(0) = y_0,\ S_3(1) = y_1$$
$$S_3'(-1) = y_{-1}',\ S_3'(1) = y_1'$$

12. 求作具有分划 $\Delta: a = x_0 < x_1 < x_2 = b$ 的三次样条 $S_3(x)$，使满足条件

$$S_3(x_i) = y_i, \quad i = 0,1,2$$

$$S_3'(x_0) = y_0', \; S_3'(x_2) = y_2'$$

第二章 数值积分

2.1 机械求积

2.1.1 求积方法的历史变迁

求积方法源于求曲边图形的面积.

古希腊数学家阿基米德(公元前287—公元前212)的重大数学成就之一是,他运用所谓穷竭法计算了一些曲边图形的面积.

譬如由抛物线 $y=x^2$ 与 $y=0$, $x=1$ 所围成的曲边三角形(如图2.1所示). 这个曲边三角形的面积 S^* 显然小于正方形 $0\le x\le 1, 0\le y\le 1$ 面积的一半. 阿基米德断言,面积 S^* 恰好等于正方形面积的三分之一,即 $S^*=\frac{1}{3}$. 在两千多年前的古代这个论断是惊人的.

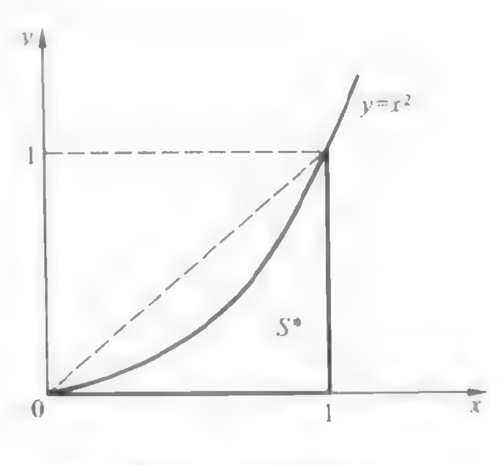


图2.1 曲边三角形

阿基米德是运用穷竭法证明这个结论的.

设将求积区间[0,1]划分为 n 等分,过分点作平行于 y 轴的直线,而将曲边三角形分割成若干窄长条,如图2.2、图2.3所示,这样得到内接与外接两个阶梯图形,当等分数 n 增大时,它们分别从内部与外部逼近抛物线 $y=x^2$.

不言而喻,将这些条状小矩形的面积累加在一起,即可获得阶梯图形的面积. 利用求和公式

$$\sum_{i=1}^{n} i^2 = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6} \tag{1}$$

易知,内、外两个阶梯图形的面积分别为

$$R_n = \frac{1}{3}\left(1 - \frac{3}{2n} + \frac{1}{2n^2}\right)$$

$$T_n = \frac{1}{3}\left(1 + \frac{3}{2n} + \frac{1}{2n^2}\right)$$

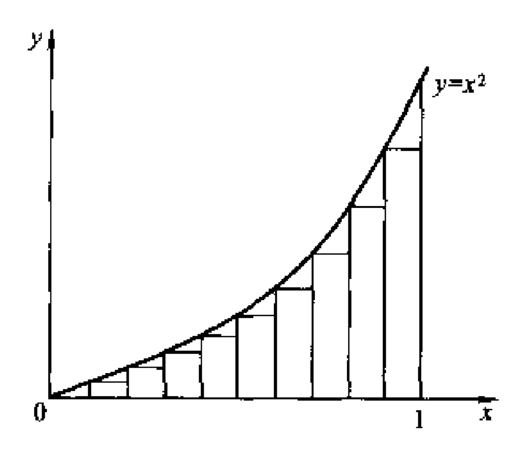


图 2.2 内接阶梯函数

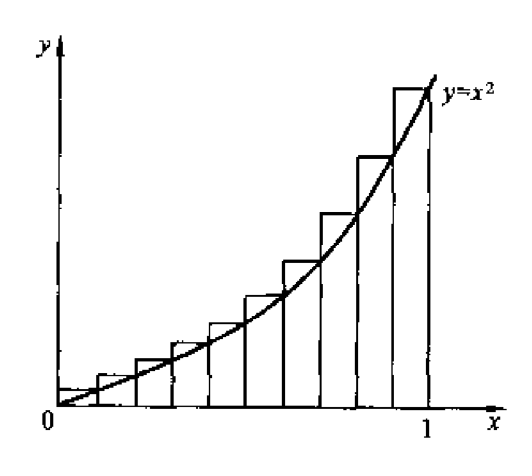


图 2.3 外接阶梯函数

显然,曲边三角形的面积 S^* 介于这两个阶梯图形的面积之间:

$$R_n < S^* < T_n$$

且只要等分数 n 足够地大,内外两个阶梯图形的面积 R_n, T_n 与值 1/3 之差值小于预先任意给定的数 ε,因之这两个阶梯图形的面积可以"穷竭"所给曲边三角形的面积.阿基米德据此断言,所给曲边三角形的面积 $S^* = \frac{1}{3}$.

在微积分方法发明之前,众多数学家依赖穷竭法求得各色各样曲边图形的

面积.穷竭法将面积计算归结为提供曲线的高度,其设计思想淳朴自然;不过,这种方法要求给出类似于式(1)的某种求和公式.而这类求和公式的建立往往是困难的.

微积分的发明使面积计算方法焕然一新.按微积分基本定理,只要提供被积函数 $f(x)$ 的原函数 $F(x)$, $F'(x)=f(x)$,便有下列求积公式:

$$\int_a^b f(x)\mathrm{d}x = F(b) - F(a)$$

由于 $y=x^2$ 的原函数是 $y=\frac{x^3}{3}$,故用微积分方法计算上述曲边三角形的面积 S^*,其全过程如下:

$$S^* = \int_0^1 x^2\mathrm{d}x = \frac{x^3}{3}\bigg|_0^1 = \frac{1}{3}$$

这就大大地简化了求积分的处理过程.

2.1.2 机械求积的概念

微积分的发明是科学史上的一项重大成就.不过微积分方法求积分也有其局限性:实际问题中碰到的被积函数 $f(x)$ 往往很复杂,找不到相应的原函数;如果 $f(x)$ 没有函数表达式,只是给出了一张数据表,则其原函数没有意义.面对这类情况,在数值求积过程中,人们又重新审视古人将积分计算归结为提供函数值的穷竭法,从而导致了所谓机械求积方法的提出.

大家知道,积分值

$$I = \int_a^b f(x)\mathrm{d}x$$

在几何上可解释为由 $x=a$, $x=b$, $y=0$ 和 $y=f(x)$ 所围成的曲边梯形的面积.积分计算之所以有困难,就是因为这个曲边梯形有一条边 $y=f(x)$ 是曲的.

依据积分中值定理,对于连续函数 $f(x)$,在 $[a,b]$ 内存在一点 ξ,成立

$$\int_a^b f(x)\mathrm{d}x = (b-a)f(\xi)$$

就是说,底为 $b-a$ 而高为 $f(\xi)$ 的矩形面积恰等于所求曲边梯形的面积 I.问题在于点 ξ 的具体位置一般是不知道的,因而难以准确地算出 $f(\xi)$ 的值.称 $f(\xi)$ 为区间 $[a,b]$ 上的**平均高度**.这样,只要对平均高度 $f(\xi)$ 提供一种算法,便相应地获得一种数值求积方法.

如果简单地选取区间 $[a,b]$ 的左、右端点或区间中点的高度作为平均高度,这样建立的求积公式分别是**左矩形公式**

$$I \approx (b-a)f(a)$$

右矩形公式

$$I\approx(b-a)f(b)$$

和中矩形公式

$$I\approx(b-a)f\left(\frac{a+b}{2}\right)$$

此外,众所周知的**梯形公式**

$$I\approx\frac{b-a}{2}[f(a)+f(b)]$$

和 **Simpson 公式**

$$I\approx\frac{b-a}{6}\left[f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right]$$

则分别可以看作用 a,b 与 $c=\frac{a+b}{2}$ 三点高度的加权平均值 $\frac{1}{2}[f(a)+f(b)]$ 和 $\frac{1}{6}[f(a)+4f(c)+f(b)]$ 作为平均高度 $f(\xi)$ 的近似值.

更一般地,取 $[a,b]$ 内若干节点 x_i 处的高度 $f(x_i)$ 通过加权平均的方法近似地得出平均高度 $f(\xi)$,这类求积方法称为**机械求积**:

$$\int_a^b f(x)\mathrm{d}x\approx(b-a)\sum_{i=0}^{n}\lambda_i f(x_i) \tag{2}$$

式中 x_i 称**求积节点**,λ_i 称**求积系数**,亦称伴随节点 x_i 的权.

这类机械求积方法直接利用某些节点上的函数值计算积分值,而将积分求值问题归结为提供函数值,这样就避开了微积分方法寻求原函数的困难.

机械求积公式的构造本质上是个选取参数 x_i,λ_i 的代数问题. 为要构造形如式(2)的求积公式,需要提供一种判定求积方法精度高低的准则.

2.1.3 求积公式的精度

机械求积方法是个近似方法. 为要保证精度,自然希望它能对"尽可能多"的简单函数是准确的. 类似于插值公式的说法(参看上一章的 1.1 节),称求积公式(2)具有 **m 阶(代数)精度**,如果它对于一切 m 次多项式是准确的,但对于 $m+1$ 次多项式不一定准确;或者说,它对于幂函数 $f(x)=x^k\ (k=0,1,\cdots,m)$ 均能准确成立,即有

$$(b-a)\sum_{i=0}^{n}\lambda_i x_i^k=\int_a^b x^k\mathrm{d}x,\quad k=0,1,\cdots,m$$

这样,机械求积公式(2)的构造问题便归结为求解如下形式的代数方程组:

$$\sum_{i=0}^{n}\lambda_i x_i^k=\frac{1}{b-a}\frac{b^{k+1}-a^{k+1}}{k+1},\quad k=0,1,\cdots,m \tag{3}$$

作为例子,对于任给两点 x_0,x_1 试构造下列机械求积公式:

$$\int_a^b f(x)\mathrm{d}x \approx (b-a)[\lambda_0 f(x_0) + \lambda_1 f(x_1)]$$

解 令它对于 $f(x)=1, f(x)=x$ 准确成立,可列出方程组

$$\begin{cases}\lambda_0 + \lambda_1 = 1\\ \lambda_0 x_0 + \lambda_1 x_1 = \dfrac{a+b}{2}\end{cases}$$

它是方程组(3)当 $m=1$ 的特殊情形,解之有

$$\lambda_0 = \frac{1}{x_1 - x_0}\left(x_1 - \frac{a+b}{2}\right)$$

$$\lambda_1 = \frac{1}{x_1 - x_0}\left(\frac{a+b}{2} - x_0\right)$$

这样设计出的求积公式具有一阶精度.特别地,若取节点 $x_0 = a, x_1 = b$,则所设计出的求积公式即为梯形公式.

本章将区分两种情况考察机械求积方法:

一种情况是,事先给定式(2)的求积节点 x_i,这时式(3)是个关于参数 λ_i 的线性方程组,处理过程相对地比较简单.

另一种情况是,令求积节点 x_i 亦自由选择,这时可显著提高求积公式(2)的精度,但由于式(3)变成关于参数 x_i, λ_i 的非线性方程组,处理过程"似乎"存在实质性的困难.

后文的2.2节、2.3节将分别考察这两类求积公式.

2.1.4 一点注记

为简化处理手续,可引进变换

$$x = \frac{b+a}{2} + \frac{b-a}{2}t$$

将求积区间$[a,b]$变到$[-1,1]$,这时积分

$$\int_a^b f(x)\mathrm{d}x = \frac{b-a}{2}\int_{-1}^1 f\left(\frac{b+a}{2} + \frac{b-a}{2}t\right)\mathrm{d}t \tag{4}$$

记 $g(t) = f\left(\dfrac{b+a}{2} + \dfrac{b-a}{2}t\right)$,则求积公式(2)变成

$$\int_{-1}^1 g(t)\mathrm{d}t \approx 2\sum_{i=0}^n \lambda_i g(t_i)$$

式中节点

$$t_i = \frac{1}{b-a}(2x_i - a - b)$$

需要注意的是,变换前后求积系数 λ_i 保持不变.而由于幂函数 $g(t) = t^k$ 在

区间[-1,1]上的积分值当 k 为奇数时恒为0,故这时方程组(3)表现为如下较为简洁的形式:

$$\sum_{i=0}^{n}\lambda_i t_i^k = \begin{cases}\dfrac{1}{k+1}, & k\ 为偶数\\ 0, & k\ 为奇数\end{cases} \tag{5}$$

这样,在设计求积公式时,不失一般性,可以着重考察区间为[-1,1]的特殊情形.

2.2 Newton - Cotes 公式

设将求积区间[a,b]划分为 n 等分,选取等分点

$$x_i = a + ih,\ h = \frac{b-a}{n},\quad i = 0,1,2,\cdots,n$$

作为求积节点构造形如式(2)的求积公式.如果这种求积公式至少有 n 阶精度,则称之为 **n 阶 Newton - Cotes 公式**.特别是,人们所熟知的梯形公式是最简单的 Newton - Cotes 公式.

2.2.1 Newton - Cotes 公式的设计方法

问题1　试以[a,b]的二等分点 $x_0 = a, x_1 = \frac{a+b}{2}, x_2 = b$ 作为求积节点构造形如

$$\int_a^b f(x)\mathrm{d}x \approx (b-a)\left[\lambda_0 f(a) + \lambda_1 f\left(\frac{a+b}{2}\right) + \lambda_2 f(b)\right]$$

的 Newton - Cotes 公式.

解　为简化处理,不妨取 $a=-1, b=1$,则上述求积公式具有形式

$$\int_{-1}^{1} f(x)\mathrm{d}x \approx 2[\lambda_0 f(-1) + \lambda_1 f(0) + \lambda_2 f(1)]$$

令它对于 $f=1, f=x, f=x^2$ 准确成立,可列出方程组

$$\begin{cases}\lambda_0 + \lambda_1 + \lambda_2 = 1\\ -\lambda_0 + \lambda_2 = 0\\ \lambda_0 + \lambda_2 = \dfrac{1}{3}\end{cases}$$

由上面第二个式子知 $\lambda_2 = \lambda_0$,这表明求积公式的内在结构具有对称性.求解上述方程组得

$$\lambda_0 = \lambda_2 = \frac{1}{6},\ \lambda_1 = \frac{2}{3}$$

容易验证这时求积公式对于 $f=x^3$ 依然是准确的,可见这样构造出的含有三个节点的 Newton-Cotes 公式

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{6}\left[f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right] \tag{6}$$

实际上有三阶精度.这是众所周知的 **Simpson 公式**.

问题 2 试以$[a,b]$的 4 等分点

$$x_i=a+ih,\ h=\frac{b-a}{4},\quad i=0,1,2,3,4$$

为节点构造形如

$$\int_a^b f(x)\mathrm{d}x \approx (b-a)[\lambda_0 f(x_0)+\lambda_1 f(x_1)+\lambda_2 f(x_2)+\lambda_3 f(x_3)+\lambda_4 f(x_4)]$$

的 Newton-Cotes 公式.

解 为简化处理,再取 $a=-1,b=1$,则求积公式具有形式

$$\int_{-1}^1 f(x)\mathrm{d}x \approx 2\left[\lambda_0 f(-1)+\lambda_1 f\left(-\frac{1}{2}\right)+\lambda_2 f(0)+\lambda_3 f\left(\frac{1}{2}\right)+\lambda_4 f(1)\right]$$

令它对于 $f=1,x,x^2,x^3,x^4$ 准确成立,可列出方程组

$$\begin{cases}\lambda_0+\lambda_1+\lambda_2+\lambda_3+\lambda_4=1\\ -\lambda_0-\frac{1}{2}\lambda_1+\frac{1}{2}\lambda_3+\lambda_4=0\\ \lambda_0+\frac{1}{4}\lambda_1+\frac{1}{4}\lambda_3+\lambda_4=\frac{1}{3}\\ -\lambda_0-\frac{1}{8}\lambda_1+\frac{1}{8}\lambda_3+\lambda_4=0\\ \lambda_0+\frac{1}{16}\lambda_1+\frac{1}{16}\lambda_3+\lambda_4=\frac{1}{5}\end{cases} \tag{7}$$

考虑到求积公式应具有对称结构,令

$$\lambda_0=\lambda_4,\ \lambda_1=\lambda_3$$

这时方程组(7)的第 2 与第 4 两个式子自然成立,而其余的式子则可化简为

$$\begin{cases}2\lambda_0+2\lambda_1+\lambda_2=1\\ 2\lambda_0+\frac{1}{2}\lambda_1=\frac{1}{3}\\ 2\lambda_0+\frac{1}{8}\lambda_1=\frac{1}{5}\end{cases}$$

据此定出

$$\lambda_0=\lambda_4=\frac{7}{90},\ \lambda_1=\lambda_3=\frac{16}{45},\ \lambda_2=\frac{2}{15}$$

这时所考察的求积公式对于 $f=x^5$ 依然能准确成立,可见这样构造出的 5 点

Newton－Cotes 公式

$$\int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{90}[7f(x_0)+32f(x_1)+12f(x_2)+32f(x_3)+7f(x_4)] \quad (8)$$

具有 5 阶精度.这一求积公式称为 **Cotes 公式**.

2.2.2 Newton－Cotes 公式的精度分析

以上事实具有普遍意义.设以 n 代表区间等分数,不难证明,当 n 为偶数时 Newton－Cotes 公式具有 $n+1$ 阶精度,这时 Newton－Cotes 公式在精度方面会获得额外的好处;而当 n 为奇数时 Newton－Cotes 公式仅有 n 阶精度.

例如,设将区间[－1,1]三等分,这时 Newton－Cotes 公式具有形式

$$\int_{-1}^1 f(x)\mathrm{d}x \approx 2\left[\lambda_0 f(-1)+\lambda_1 f\left(-\frac{1}{3}\right)+\lambda_2 f\left(\frac{1}{3}\right)+\lambda_3 f(1)\right]$$

令它对于 $f=1,x,x^2,x^3$ 准确成立,可列出方程组

$$\begin{cases}\lambda_0+\lambda_1+\lambda_2+\lambda_3=1\\ -\lambda_0-\frac{1}{3}\lambda_1+\frac{1}{3}\lambda_2+\lambda_3=0\\ \lambda_0+\frac{1}{9}\lambda_1+\frac{1}{9}\lambda_2+\lambda_3=\frac{1}{3}\\ -\lambda_0-\frac{1}{27}\lambda_1+\frac{1}{27}\lambda_2+\lambda_3=0\end{cases}$$

考虑到求积公式内在结构的对称性,令

$$\lambda_0=\lambda_3,\ \lambda_1=\lambda_2$$

则上面的第二和第 4 两个式子自然成立,从而方程组可化简为

$$\begin{cases}\lambda_0+\lambda_1=\frac{1}{2}\\ \lambda_0+\frac{1}{9}\lambda_1=\frac{1}{6}\end{cases}$$

据此定出

$$\lambda_0=\lambda_3=\frac{1}{8},\ \lambda_1=\lambda_2=\frac{3}{8}$$

这样设计出的 4 点 Newton－Cotes 公式

$$\int_{-1}^1 f(x)\mathrm{d}x \approx \frac{1}{4}\left[f(-1)+3f\left(-\frac{1}{3}\right)+3f\left(\frac{1}{3}\right)+f(1)\right] \quad (9)$$

对于 $f=x^4$ 不准确,可见它仅有三阶精度,即同三点 Newton－Cotes 公式——Simpson 公式精度相当.

数值算例同样说明了这个事实.

例 1 用 Newton－Cotes 公式计算积分

$$I=\int_0^1\frac{\sin x}{x}\mathrm{d}x$$

解 计算结果见表2.1.表中 n 表示区间等分数，I_n 为相应的积分值，最末一列指明有效数字的位数 m（I 的准确值为0.946 083 1）.

表 2.1

n	I_n	m	n	I_n	m
1	0.927 035 4	1	4	0.946 083 0	6
2	0.946 135 9	3	5	0.946 083 0	6
3	0.946 110 9	3			

从表2.1的计算结果确实可以看到这样的事实：在Newton–Cotes公式中，二等分与三等分的求积公式精度相当，4等分与5等分的求积公式也是如此.

如果进一步增加区间等分数，那么，所设计出的求积公式由于稳定性差而没有实用价值.因之，在众多的Newton–Cotes型求积公式中，人们更感兴趣的是梯形公式（它最简单、最基本）、Simpson公式（$n=2$）与Cotes公式（$n=4$）.

2.3 Gauss公式

上一节在构造Newton–Cotes公式时，限定用积分区间的等分点作为求积节点，这样做简化了处理过程（所归结出的代数方程组是线性的），但同时限制了精度.如果求积节点可以自由选择，则求积公式(2)中含有 $2n+2$ 个待定参数 x_i 与 λ_i，$i=0,1,\cdots,n$，适当选取这些参数可以使求积公式具有 $2n+1$ 阶精度.这种高精度的求积公式称作 **Gauss公式**.

首先取积分区间为$[-1,1]$而考察如下形式的求积公式：

$$\int_{-1}^{1}f(x)\mathrm{d}x\approx 2\sum_{i=0}^{n}\lambda_i f(x_i)$$

为使它成为Gauss型的，只要令其参数 x_i、λ_i 满足 $m=2n+1$ 的代数方程组(5)，即

$$\sum_{i=0}^{n}\lambda_i x_i^k=\frac{1+(-1)^k}{2(k+1)},\quad k=0,1,\cdots,2n+1 \tag{10}$$

特别地，对于一点Gauss公式（$n=0$）：

$$\int_{-1}^{1}f(x)\mathrm{d}x\approx 2\lambda_0 f(x_0)$$

令对 $f(x)=1$，$f(x)=x$ 准确成立，有

$$\begin{cases}\lambda_0 = 1 \\ \lambda_0 x_0 = 0\end{cases}$$

据此定出 $\lambda_0 = 1, x_0 = 0$. 可见**一点 Gauss 公式**是人们所熟知的中矩形公式

$$G_1 = 2f(0) \tag{11}$$

它具有一阶精度,即一点 Gauss 公式与两点 Newton - Cotes 公式(梯形公式)的精度相当.

再考察两点 Gauss 公式($n = 1$)

$$\int_{-1}^{1} f(x)\mathrm{d}x \approx 2[\lambda_0 f(x_0) + \lambda_1 f(x_1)] \tag{12}$$

令它对于 $f = 1, x, x^2, x^3$ 准确成立,有

$$\begin{cases}\lambda_0 + \lambda_1 = 1 \\ \lambda_0 x_0 + \lambda_1 x_1 = 0 \\ \lambda_0 x_0^2 + \lambda_1 x_1^2 = \dfrac{1}{3} \\ \lambda_0 x_0^3 + \lambda_1 x_1^3 = 0\end{cases} \tag{13}$$

这样归结出的方程组是方程组(10)取 $n = 1$ 的特殊情形. 方程组(13)是个含有 4 个未知数的非线性方程组,它的求解似乎有实质性的困难.

可以运用对称性原则进行简化处理. Gauss 公式具有高精度,它的结构应当具有鲜明的对称性. 特别地,对于两点 Gauss 公式(12),令

$$\lambda_1 = \lambda_0, \quad x_1 = -x_0$$

则方程组(13)的第 2 与第 4 两个式子自然成立,因而可将它化简为

$$\begin{cases}2\lambda_0 = 1 \\ 2\lambda_0 x_0^2 = \dfrac{1}{3}\end{cases}$$

由此即得

$$\lambda_0 = \lambda_1 = \frac{1}{2}, \quad x_1 = -x_0 = \frac{1}{\sqrt{3}}$$

这样构造出的**两点 Gauss 公式**

$$G_2 = f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right) \tag{14}$$

具有三阶精度,即两点 Gauss 公式与三点 Newton - Cotes 公式(Simpson 公式)的精度相当.

进一步考察三点 Gauss 公式

$$\int_{-1}^{1} f(x)\mathrm{d}x \approx 2[\lambda_0 f(x_0) + \lambda_1 f(x_1) + \lambda_2 f(x_2)]$$

为使它具有5阶精度,考察 $n=2$ 的方程组(10):

$$\begin{cases}\lambda_0+\lambda_1+\lambda_2=1\\ \lambda_0x_0+\lambda_1x_1+\lambda_2x_2=0\\ \lambda_0x_0^2+\lambda_1x_1^2+\lambda_2x_2^2=\dfrac{1}{3}\\ \lambda_0x_0^3+\lambda_1x_1^3+\lambda_2x_2^3=0\\ \lambda_0x_0^4+\lambda_1x_1^4+\lambda_2x_2^4=\dfrac{1}{5}\\ \lambda_0x_0^5+\lambda_1x_1^5+\lambda_2x_2^5=0\end{cases}$$

这是一个相当复杂的非线性方程组,仍运用对称性原则,令

$$x_2=-x_0,\quad x_1=0,\quad \lambda_2=\lambda_0$$

则可将上述方程组化简为

$$\begin{cases}2\lambda_0+\lambda_1=1\\ 2\lambda_0x_0^2=\dfrac{1}{3}\\ 2\lambda_0x_0^4=\dfrac{1}{5}\end{cases}$$

据此容易定出

$$x_2=-x_0=\sqrt{\frac{3}{5}},\quad x_1=0$$

$$\lambda_2=\lambda_0=\frac{5}{18},\qquad \lambda_1=\frac{4}{9}$$

这样构造出的**三点 Gauss 公式**是

$$G_3=\frac{5}{9}f\left(-\sqrt{\frac{3}{5}}\right)+\frac{8}{9}f(0)+\frac{5}{9}f\left(\sqrt{\frac{3}{5}}\right)\tag{15}$$

它具有5阶精度,即其精度与5点 Newton－Cotes 公式(Cotes 公式)相当.

不言而喻,更高阶的 Gauss 公式的构造更为复杂,其实有实用价值的仅仅是上述几个低阶 Gauss 公式.

如果积分区间为$[a,b]$,依式(4),则一点、二点和三点 Gauss 公式分别为

$$G_1=(b-a)f\left(\frac{a+b}{2}\right)$$

$$G_2=\frac{b-a}{2}\left[f\left(\frac{b+a}{2}-\frac{b-a}{2\sqrt{3}}\right)+f\left(\frac{b+a}{2}+\frac{b-a}{2\sqrt{3}}\right)\right]$$

$$G_3=\frac{b-a}{2}\left[\frac{5}{9}f\left(\frac{b+a}{2}-\sqrt{\frac{3}{5}}\frac{b-a}{2}\right)+\frac{8}{9}f\left(\frac{b+a}{2}\right)+\frac{5}{9}f\left(\frac{b+a}{2}+\sqrt{\frac{3}{5}}\frac{b-a}{2}\right)\right]$$

2.4 复化求积法

2.4.1 复化求积公式

设将求积区间$[a,b]$划分为n等分，步长$h=\frac{b-a}{n}$，分点为$x_i=a+ih$，$i=0,1,\cdots,n$. 所谓**复化求积法**，就是先用低阶求积公式求得每个子段$[x_i,x_{i+1}]$上的积分值I_i，然后再将它们累加求和，用各段积分之和$\sum_{i=0}^{n-1}I_i$作为所求积分的近似值.

复化求积法对于人们其实并不陌生. 前面已指出，早在两千多年前，古希腊的阿基米德已经运用复化矩形公式计算曲边图形的面积(参看2.1.1小节).

下面具体列出复化求积的计算公式.

复化梯形公式是

$$T_n=\sum_{i=0}^{n-1}\frac{h}{2}[f(x_i)+f(x_{i+1})]$$

$$=\frac{b-a}{2n}\left[f(a)+2\sum_{i=1}^{n-1}f(x_i)+f(b)\right] \tag{16}$$

记子段$[x_i,x_{i+1}]$的中点为$x_{i+\frac{1}{2}}$，则**复化 Simpson 公式**为

$$S_n=\sum_{i=0}^{n-1}\frac{h}{6}[f(x_i)+4f(x_{i+\frac{1}{2}})+f(x_{i+1})]$$

$$=\frac{b-a}{6n}\left[f(a)+4\sum_{i=0}^{n-1}f(x_{i+\frac{1}{2}})+2\sum_{i=1}^{n-1}f(x_i)+f(b)\right] \tag{17}$$

如果将每个子段$[x_i,x_{i+1}]$分为4等分，内分点依次记为$x_{i+\frac{1}{4}},x_{i+\frac{1}{2}},x_{i+\frac{3}{4}}$，则**复化 Cotes 公式**为

$$C_n=\frac{b-a}{90n}\left[7f(a)+32\sum_{i=0}^{n-1}f(x_{i+\frac{1}{4}})+14\sum_{i=0}^{n-1}f(x_{i+\frac{1}{2}})+\right.$$

$$\left.32\sum_{i=0}^{n-1}f(x_{i+\frac{3}{4}})+7f(b)\right] \tag{18}$$

例2 用函数$f(x)=\frac{\sin x}{x}$的数据表(表2.2)计算积分

$$I=\int_0^1\frac{\sin x}{x}\mathrm{d}x$$

表 2.2

x	$f(x)$	x	$f(x)$
0	1.000 000 0	5/8	0.936 155 6
1/8	0.997 397 8	3/4	0.908 851 6
1/4	0.989 615 8	7/8	0.877 192 5
3/8	0.976 726 7	1	0.841 470 9
1/2	0.958 851 0		

解　判定一种算法的优劣，计算量是一个重要的因素，由于在求 $f(x)$ 的函数值时，通常要做许多次加减乘除四则运算，因此在统计求积公式 $\sum_i \lambda_i f(x_i)$ 的计算量时，只要统计求函数值 $f(x_i)$ 的次数.

用复化求积法求例 2 的积分值. 取 $n=8$ 用复化梯形公式(16)求得

$$T_8=0.945\ 690\ 9$$

再取 $n=4$ 用复化 Simpson 公式(17)得

$$S_4=0.946\ 083\ 2$$

比较这两个结果，它们都需要提供 9 个点上的函数值，工作量基本相同，然而精度却差别很大，同积分的准确值 0.946 083 1 比较，复化梯形法的结果 T_8 只有 2 位有效数字，而复化 Simpson 法的结果 S_4 却有 6 位有效数字. 这个例子再一次表明选择合适的算法意义重大.

2.4.2　变步长的梯形法

这里所面对的问题是，运用某种复化求积方法可以获得积分值 I 的近似值 $I(h)$，而所求积分值 I 则视为 $I(h)$ 当 $h\to 0$ 时的极限值. 这样，只要步长 h 足够小，即可取 $I(h)$ 作为所求积分值 I.

问题在于如何选取合适的步长 h？步长过大精度不能保证，步长太小则会导致计算量的显著增加，选择步长需要在精度与计算量两者之间实现合理的平衡，然而事先给出一个合适的步长通常是困难的.

实际计算时，希望在保证精度的前提下选取尽可能大的步长，为此常常采取如下策略：事先预报某个步长 h（可适当放大一点，以留有余地），然后将步长逐次减半，直到二分前后两个近似值的偏差 $\left|I\left(\frac{h}{2}\right)-I(h)\right|$ 在精度范围内可以忽略为止. 这种在计算过程中自选步长的方法称作**变步长方法**.

现在在变步长的过程中探讨梯形法的计算规律. 设将积分区间分为 n 等分，则一共有 $n+1$ 个分点

$$x_i = a + ih,\ h = \frac{b-a}{n},\quad i = 0,1,\cdots,n$$

先考察一个子段$[x_i, x_{i+1}]$,其中点$x_{i+\frac{1}{2}} = a + \left(i+\frac{1}{2}\right)h$,该子段上二分前后两个梯形值

$$T_1 = \frac{h}{2}[f(x_i) + f(x_{i+1})]$$

$$T_2 = \frac{h}{4}[f(x_i) + 2f(x_{i+\frac{1}{2}}) + f(x_{i+1})]$$

显然有下列关系

$$T_2 = \frac{1}{2}T_1 + \frac{h}{2}f(x_{i+\frac{1}{2}})$$

将这一关系式关于 i 从 0 到 $n-1$ 累加求和,即可导出如下递推算式

$$T_{2n} = \frac{1}{2}T_n + \frac{h}{2}\sum_{i=0}^{n-1}f(x_{i+\frac{1}{2}}) \tag{19}$$

式中 $h = \frac{b-a}{n}$ 为二分前的步长,$x_{i+\frac{1}{2}} = a + \left(i+\frac{1}{2}\right)h$.

图 2.4 是变步长梯形法的算法框图.其中 T_1 和 T_2 分别代表二分前后的积分值,各框的具体含义是:

[框 1] 准备初值.

[框 2] 按式(19)求二分后的梯形值.

从第 1 个分点 $x = a + \frac{h}{2}$ 出发,取 h 为步长逐步向右跨,即可依次确定式(19)中每个分点.图 2.4 将所得到的分点暂存于单元 x 中.

[框 3] 控制精度.

[框 4] 修改步长.

最终取二分后的积分值 T_2 作为计算结果.

例 3 用变步长梯形法计算积分

$$I = \int_0^1 \frac{\sin x}{x}\mathrm{d}x$$

解 先对整个区间[0,1]用梯形公式.对于被积函数 $f(x) = \frac{\sin x}{x}$,由于 $f(0) = 1$,$f(1) = 0.841\ 470\ 9$,故

$$T_1 = \frac{1}{2}[f(0) + f(1)] = 0.920\ 735\ 5$$

然后将区间二分,由于 $f\left(\frac{1}{2}\right) = 0.958\ 851\ 0$ 利用递推公式(19)得

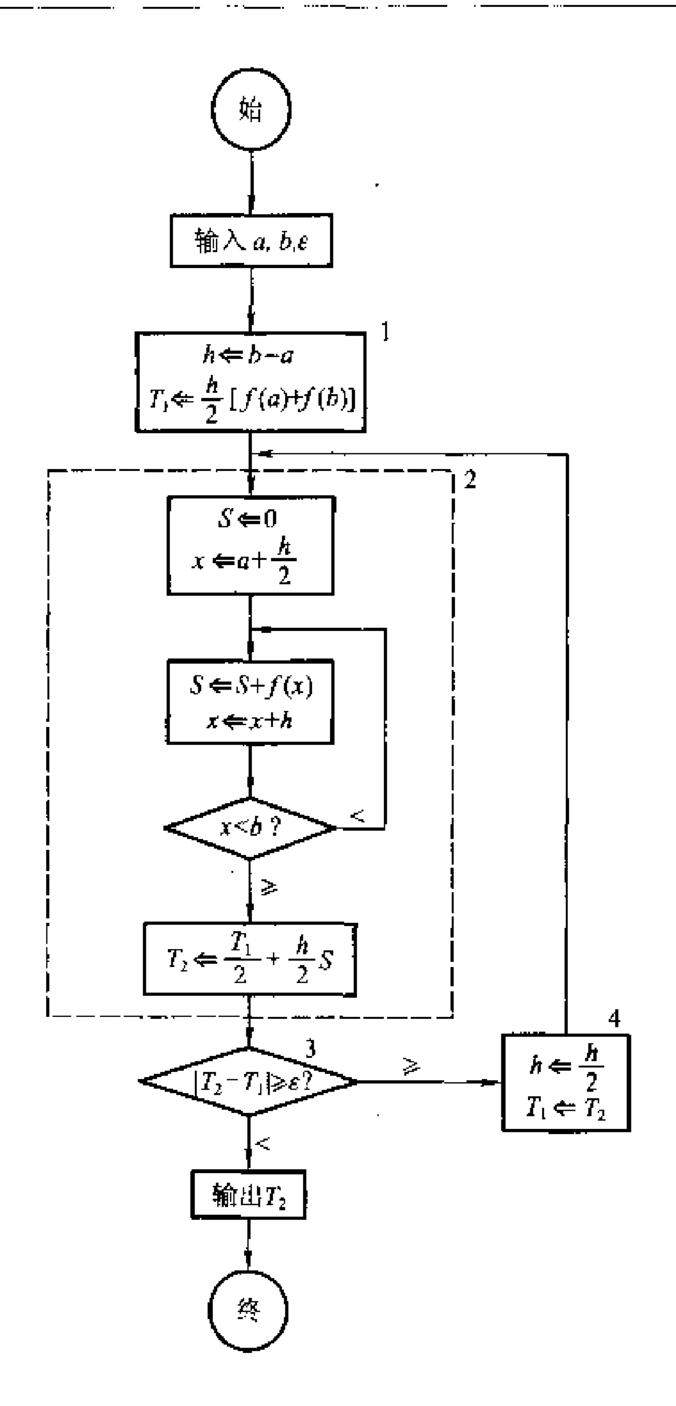


图 2.4 变步长梯形法的计算流程

$$T_2=\frac{1}{2}T_1+\frac{1}{2}f\left(\frac{1}{2}\right)=0.939\ 793\ 3$$

再二分一次，并计算新分点上的函数值

$$f\left(\frac{1}{4}\right)=0.989\ 615\ 8$$

$$f\left(\frac{3}{4}\right)=0.908\ 851\ 6$$

再用式(19)求得

$$T_4 = \frac{1}{2}T_2 + \frac{1}{4}\left[f\left(\frac{1}{4}\right) + f\left(\frac{3}{4}\right)\right]$$
$$= 0.944\,513\,5$$

这样不断二分下去,计算结果见表 2.3(表中 k 代表二分次数,区间等分数 $n = 2^k$).这里,用变步长方法二分 10 次得到了有 7 位有效数字的积分值 $I = 0.946\,083\,1$.

表 2.3

k	T	k	T
0	0.920 735 5	6	0.946 076 9
1	0.939 793 3	7	0.946 081 5
2	0.944 513 5	8	0.946 082 7
3	0.945 690 9	9	0.946 083 0
4	0.945 985 0	10	0.946 083 1
5	0.946 059 6		

2.5 Romberg 加速算法

2.5.1 梯形法的加速

复化梯形法的算法简单,但精度低,收敛速度缓慢.能否设法加工梯形值以提高精度呢?

考察二分前后的梯形值

$$T_1 = \frac{b-a}{2}[f(a) + f(b)]$$

$$T_2 = \frac{b-a}{4}[f(a) + 2f(c) + f(b)], \quad c = \frac{a+b}{2}$$

它们都只有一阶精度.现将两者进行松弛,令

$$S_1 = (1+\omega)T_2 - \omega T_1$$
$$= T_2 + \omega(T_2 - T_1) \tag{20}$$

显然,不管因子 ω 如何选择,松弛公式(20)均具有一阶精度.注意到节点 $c = \frac{a+b}{2}$ 是二等分点,为使它具有二阶精度,它必须是二等分的 Newton-Cotes 公式——即 Simpson 公式.因之,这个问题可表述为,能否找到合适的松弛因子 ω,

使二分前后的两个梯形值 T_1, T_2 按式(20)松弛生成 Simpson 值 S_1. 注意到

$$S_1 = \frac{b-a}{6}[f(a) + 4f(c) + f(b)]$$

比较式(20)两端 $f(a), f(c)$ 与 $f(b)$ 的系数,容易定出

$$\omega = \frac{1}{3}$$

从而有

$$S_1 = \frac{4}{3}T_2 - \frac{1}{3}T_1$$

其复化形式为

$$S_n = \frac{4}{3}T_{2n} - \frac{1}{3}T_n \tag{21}$$

这就是说,用二分前后的两个梯形值 T_n 与 T_{2n} 按式(21)进行加工,结果生成 Simpson 值 S_n.

2.5.2 Simpson 法再加速

进一步加工 Simpson 值. 将区间 $[a, b]$ 分为 4 等分,分点 $x_i = a + i\frac{b-a}{4}$, $i = 0, 1, \cdots, 4$, 则二分前后的 Simpson 值 S_1, S_2 都具有三阶精度. 适当选取因子 ω, 以将松弛值

$$C_1 = (1+\omega)S_2 - \omega S_1 \tag{22}$$

提高到 4 阶精度. 注意到这里节点 $x_i, i = 0, 1, \cdots, 4$ 是 4 等分点,这样设计的求积公式(22)应当是 Cotes 公式

$$C_1 = \frac{b-a}{90}[7f(x_0) + 32f(x_1) + 12f(x_2) + 32f(x_3) + 7f(x_4)]$$

比较式(22)两端 $f(x_i), i = 0, 1, \cdots, 4$ 的系数可定出

$$\omega = \frac{1}{15}$$

从而有

$$C_n = \frac{16}{15}S_{2n} - \frac{1}{15}S_n \tag{23}$$

这样,将二分前后的两个 Simpson 值进行再加工,可进一步生成 Cotes 值.

2.5.3 Cotes 法的进一步加速

再加工 Cotes 值. 将积分区间 $[a, b]$ 分为 8 等分,分点 $x_i = a + i\frac{b-a}{8}$, $i = 0, 1, \cdots, 8$, 则二分前后的 Cotes 值为

$$C_1=\frac{b-a}{90}[7f(x_0)+32f(x_2)+12f(x_4)+32f(x_6)+7f(x_8)]$$

$$C_2=\frac{b-a}{180}[7f(x_0)+32f(x_1)+12f(x_2)+32f(x_3)+14f(x_4)+$$
$$32f(x_5)+12f(x_6)+32f(x_7)+7f(x_8)]$$

这时松弛公式

$$R_1=(1+\omega)C_2-\omega C_1$$

至少有 5 阶精度.现在选取合适的松弛值 ω 将以上求积公式提高到 6 阶精度,为此令它对于 $f(x)=x^6$ 准确成立,据此定出

$$\omega=\frac{1}{63}$$

这样设计出的求积公式

$$R_n=\frac{64}{63}C_{2n}-\frac{1}{63}C_n \tag{24}$$

称作 **Romberg 公式**.可以验证它实际上有 7 阶精度.注意到 Romberg 公式的求积节点是区间 $[a,b]$ 的 8 等分点,但仅能保证它有 7 阶精度,因此它不再属于 Newton – Cotes 公式的范畴.

2.5.4 Romberg 算法的计算流程

在步长二分的过程中运用公式(21),(23),(24)加工三次,就能将粗糙的梯形值 T_n 逐步加工成高精度的 Romberg 值 R_n,或者说,将收敛缓慢的梯形值序列 $\{T_n\}$ 加工成收敛迅速的 Romberg 值序列 $\{R_n\}$.这种加速算法称作 **Romberg 算法**,其加速过程如图 2.5 所示.图 2.6 描述 Romberg 算法的计算流程.

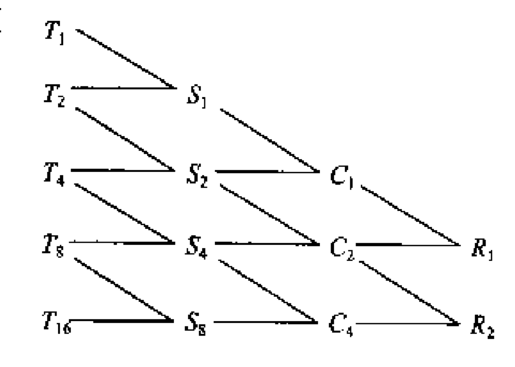


图 2.5 Romberg 算法的加速过程

例 4 用 Romberg 算法加工表 2.3 的梯形值,计算结果见表 2.4,表中 k 代表二分次数.

表 2.4

k	T_{2^k}	$S_{2^{k-1}}$	$C_{2^{k-2}}$	$R_{2^{k-3}}$
0	0.920 735 5			
1	0.939 793 3	0.946 145 9		
2	0.944 513 5	0.946 086 9	0.946 083 0	
3	0.945 690 9	0.946 083 4	0.946 083 1	0.946 083 1

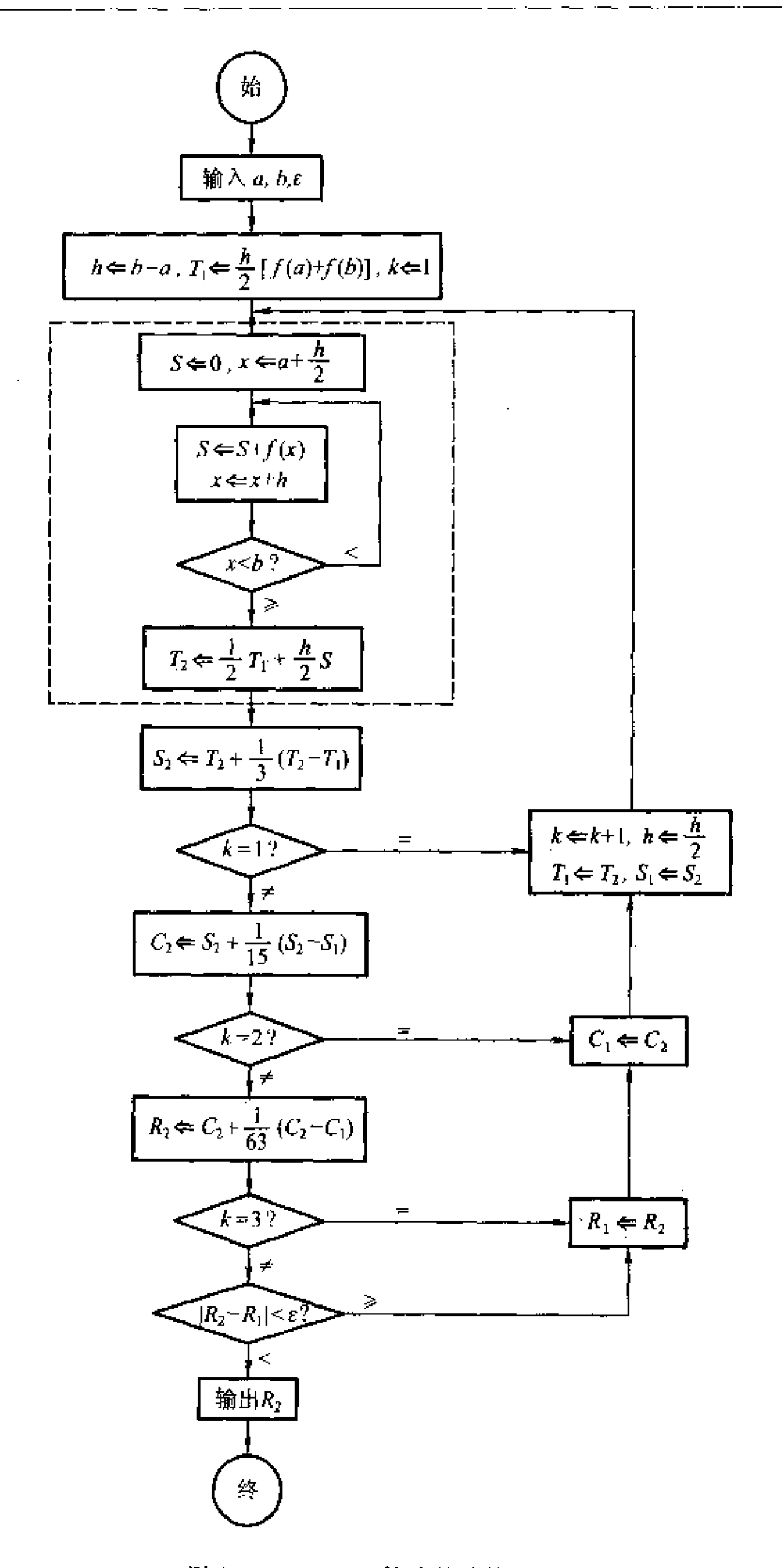


图 2.6 Romberg 算法的计算流程图

这里用二分三次的数据(它们的精度都很低,只有2、3位有效数字),通过三次加速获得了例3需要二分10次才能求得的结果,而加速过程的计算量可以忽略不计,可见Romberg算法的加速效果是极其显著的.Romberg算法是个优秀算法.

2.6 数值微分

微分和积分是一对互逆的数学运算.下面将参照前述数值积分来讨论数值微分.

2.6.1 数值求导的差商公式

按照数值分析的定义,导数 $f'(a)$ 是差商 $\frac{f(a+h)-f(a)}{h}$ 当 $h\to 0$ 时的极限,因之,如果精度要求不高,可以简单地取差商作为导数的近似值,这样便建立起一种求导公式,称为**向前差商**

$$f'(a)\approx\frac{f(a+h)-f(a)}{h}$$

类似地,亦可用**向后差商**

$$f'(a)\approx\frac{f(a)-f(a-h)}{h}$$

或用**中心差商**

$$f'(a)\approx\frac{f(a+h)-f(a-h)}{2h}$$

后一种求导方法亦称**中点方法**,它是前两种方法的算术平均.

在图形上(参看图2.7),上述三种导数的近似值分别表示弦线 AB、AC 和 BC 的斜率.比较这三条弦线与切线 AT(其斜率等于导数值 $f'(a)$)平行的程度,从图形上可以明显地看出,其中以 BC 的斜率更接近于切线 AT 的斜率.因此就精度而言,以中点方法更为可取.

为要利用中点公式

$$G(h)=\frac{f(a+h)-f(a-h)}{2h}$$

计算导数值 $f'(a)$,首先必须选取合适的步长.步长太大精度难以保证,步长太小又会导致舍入误差的增长,在实际计算过程中,希望在保证精度的前提下选取尽可能大的步长,然而事先给出一个合适的步长往往是困难的.通常在步长二分的变步长过程中实现步长的自动选择.

例5 用变步长的中点方法求 e^x 在 $x=1$ 的导数值,设取 $h=0.8$ 起算.

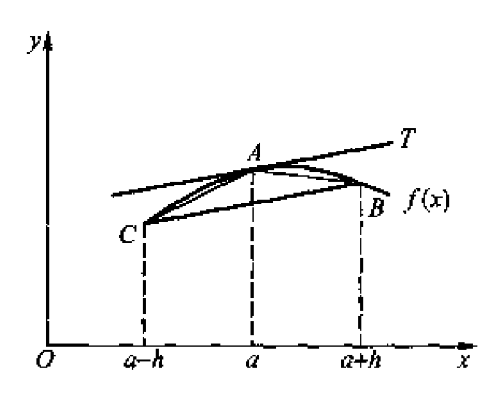


图 2.7　三种差商值的比较

解　这里采用的计算公式是

$$G(h)=\frac{e^{1+h}-e^{1-h}}{2h}$$

计算结果见表2.5,表中 k 代表二分的步数,步长 $h=\frac{0.8}{2^k}$.可以看到,二分9次得出结果 $G=2.718\ 28$,它的每一位数字都是有效数字(所求导数的精确值为 $e=2.718\ 281\ 8$).

表 2.5

k	$G(h)$	k	$G(h)$
0	3.017 65	6	2.718 35
1	2.791 35	7	2.718 30
2	2.736 44	8	2.718 29
3	2.722 81	9	2.718 28
4	2.719 41	10	2.718 28
5	2.718 56		

2.6.2　数值求导公式的设计方法

同数值求积一样,所谓数值求导,就是将导数计算归结为提供若干节点上的函数值.前述几种差商公式都是特殊的数值求导公式.

设已知 $f(x)$在一组节点 $x_i=x_0+ih,i=0,1,\cdots,n$ 上的函数值 $f(x_i)$,用这些数据组合生成给定节点 x_k 的导数值,即令求导公式具有形式

$$f'(x_k)\approx\frac{1}{h}\sum_{i=0}^{n}\lambda_i f(x_i) \qquad (25)$$

很自然,为使求导公式具有足够的精度,要求它对于足够高次的多项式能准确成立,这又归结为求解关于参数 λ_i 的代数方程组.例如,试对于给定的三个节点 $x_0, x_1 = x_0 + h, x_2 = x_0 + 2h$ 构造求导公式

$$f'(x_0) \approx \frac{1}{h}[\lambda_0 f(x_0) + \lambda_1 f(x_1) + \lambda_2 f(x_2)]$$

令它对于 $f(x) = 1, x, x^2$ 准确成立,可列出方程

$$\begin{cases} \lambda_0 + \lambda_1 + \lambda_2 = 0 \\ \lambda_0 x_0 + \lambda_1 x_1 + \lambda_2 x_2 = h \\ \lambda_0 x_0^2 + \lambda_1 x_1^2 + \lambda_2 x_2^2 = 2x_0 h \end{cases}$$

为简化计算,不妨令 $x_0 = 0, h = 1$,则方程组化简为

$$\begin{cases} \lambda_0 + \lambda_1 + \lambda_2 = 0 \\ \lambda_1 + 2\lambda_2 = 1 \\ \lambda_1 + 4\lambda_2 = 0 \end{cases}$$

据此解出

$$\lambda_0 = -\frac{3}{2}, \quad \lambda_1 = 2, \quad \lambda_2 = -\frac{1}{2}$$

这样设计出的求导公式是

$$f'(x_0) \approx \frac{1}{2h}[-3f(x_0) + 4f(x_1) - f(x_2)] \tag{26}$$

亦可基于插值方法构造求导公式.设已知 $f(x)$ 在节点 $x_i = x_0 + ih, i = 0, 1, \cdots, n$ 的函数值 $f(x_i)$,作 n 次插值多项式 $p_n(x)$,并取 $p_n'(x_k)$ 的值作为给定节点 x_k 处的导数值 $f'(x_k)$,这样建立起来的求导公式同样具有式(25)的形式.

譬如,利用给定的三个节点 $x_0, x_1 = x_0 + h, x_2 = x_0 + 2h$ 上的函数值作插值多项式

$$p_2(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} f(x_0) + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} f(x_1) + \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} f(x_2) \tag{27}$$

两端求导并令 $f'(x_0) \approx p_2'(x_0)$,那么,据此同样可以导出公式(26).

用插值多项式 $p_n(x)$ 作为 $f(x)$ 的近似函数,还可以建立高阶数值微分公式

$$f^{(k)}(x) \approx p_n^{(k)}(x)$$

例如,将式(27)求导两次,即可导出二阶求导公式

$$f''(x_1) \approx \frac{1}{h^2}[f(x_0) - 2f(x_1) + f(x_2)]$$

2.7 千古绝技"割圆术"

圆,它是人们最为熟悉,应用最为广泛的一种几何曲线.

千百年来,人类对圆进行过长期深入的观察与分析.古人早就知道圆中有个数字不变量:不管圆的大小如何,其周长与直径的比,以及其面积除以半径的平方,是同一个定数.这个数学常数称为**圆周率**,记做π.

虽然人们早就知道圆周率是个定数,但它的精确计算却是数学史上一道千古难题.

2.7.1 "缀术"之谜

上古用"3"规定圆周率,突出人们对圆周率这个奇妙数字的宠爱与崇拜."周三径一",许多古代经典——包括东方的《易经》与西方的《圣经》,都规定了这个教条.

据文献记载,首先实现圆周率精确计算的是古希腊的阿基米德.早在公元前3世纪,他用内接与外切正96边形逼近圆周,获得π的近似值3.14.

中国数学家对于圆周率计算也做出了杰出的贡献.

公元3世纪,魏晋大数学家刘徽用正3 072边形逼近圆周,获得π的近似值3.141 6.这个结果准确到小数点第4位.在生产力低下的古代,如此高的精度对于实际应用已是绰绰有余了.

事隔两百年之后,公元5世纪,南北朝祖冲之更进一步获得了准确到小数点7位的圆周率3.141 592 6.这是一项千年称雄的数学成就.

祖冲之称其算法为"缀术".但缀术已千年失传,致使祖冲之计算圆周率的奇妙算法成了数学史上一桩千古疑案.

2.7.2 奇妙的"割圆术"

公元263年,魏晋刘徽为中华算经《九章算术》作注.刘徽的《九章算术注》奠定了中华数学的理论基础.

关于圆面积计算,《九章算术》的"圆田术"指出:"半周半径相乘得积步".这就是说,圆面积等于半周长与半径的乘积.刘徽写了一篇注记附于其后.这篇"圆田术"的刘徽注后人称为《割圆术》.

《割圆术》全文约1 800字.其内容翔实,结构紧凑,气势磅礴而寓意深邃,是数学史上一篇千古奇文.

割圆术的主要内容是基于圆面积计算设计圆周率的高效算法.这一算法包含细分逼近、递推计算与松弛加速三个环节.

1. 细分逼近的二分策略

刘徽从圆周的6等分做起,反复二分各个弧段,逐步将圆周分割成12分,24分,48分,…. 在弧段二分过程中,等分数逐步倍增,二分 k 次圆周被分割成 6×2^k 个弧段,或者说,圆被切割成 6×2^k 个小扇形片(见图2.8).

刘徽在弧段二分的过程中考察了小扇形的近似计算问题,他采取"以直代曲"的逼近策略,将小扇形转化为小三角形. 这些小三角形合并为圆的内接正多边形(图2.8). 二分 k 次后圆内接正多边形的边数为 6×2^k.

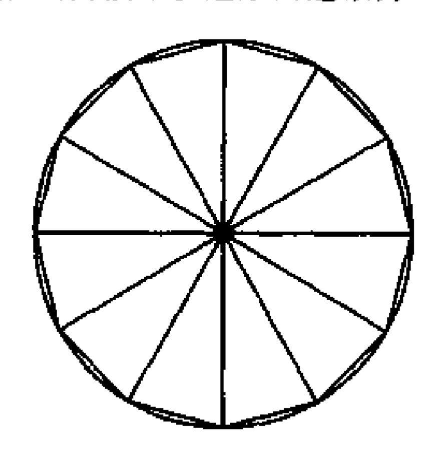


图2.8 二分割圆

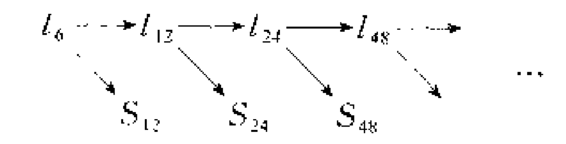
2. 递推计算的刘徽公式

刘徽具体给出了二分前后从内接正 n 边形到 $2n$ 边形的递推公式. 设圆半径为 r,内接正 n 边形的边长为 l_n,面积为 S_n,刘徽利用勾股定理导出了如下形式的递推公式

$$l_{2n}=\sqrt{\left(\frac{l_n}{2}\right)^2+\left[r-\sqrt{r^2-\left(\frac{l_n}{2}\right)^2}\right]^2}$$

$$S_{2n}=\frac{n}{2}rl_n$$

他据此设计出一个完整的迭代算法,其计算流程如下:



刘徽在《割圆术》一文中,用了三分之二的篇幅,详尽地记录了割圆计算的二分过程,记录了计算过程中每一个中间数据. 割圆计算是枯燥无味的,它是简单计算的重复. 通过这种不厌其烦的重复,刘徽深刻揭示了算法设计的基本理念:

简单的重复生成复杂.

3. 数据松弛的加速技术

刘徽深入地考察了数据的精加工问题.他发现,如果选取松弛因子

$$\omega=\frac{36}{105}$$

则可将两个粗糙数据 S_{96} 与 S_{192} 加工成高精度的结果 $S_{3\,072}$.

为便于进行筹算,刘徽令 $r=10$,这时圆面积 $S^*=100\pi$.刘徽通过二分割圆手续求得

$$S_{96}=313\frac{584}{625},\quad S_{192}=314\frac{64}{625}$$

这两个数据的精度都很差,相当于圆周率 3.14.其实阿基米德早已掌握这些数据.刘徽比阿基米德技高一筹,他设计出下列数据加工过程

$$\begin{aligned}&S_{192}+\frac{36}{105}(S_{192}-S_{96})\\&=314\frac{64}{625}+\frac{36}{105}\times\frac{105}{625}\\&=314\frac{100}{625}=314\frac{4}{25}\end{aligned}$$

据此获得高精度的圆周率 3.141 6.刘徽强调指出,这样加工出的结果是 $S_{3\,072}$,即有

$$S_{3\,072}\approx S_{192}+\omega(S_{192}-S_{96})$$

式中

$$\omega=\frac{36}{105}$$

在这里,刘徽利用两个粗糙的数据通过松弛方法加工出高精度的结果,而其加工过程几乎不耗费计算量,这是一种加速算法.

众所周知,在西方,数值求积的 Romberg 算法(参看 2.5 节)直到 1955 年才被发现.人们普遍认为这一算法开创了加速算法设计的先河.基于余项展开的快速算法设计被认为是计算数学中一个突出的难点.令人难以置信的是,古代中国数学家早在 1 700 多年以前已经掌握了这门绝技,这是超越时代的大智慧.

小 结

1. 本章首先提出了机械求积的概念.众所周知,基于 Newton－Leibniz 公式,微积分学将积分计算归结为寻求所谓"原函数".问题在于科学计算中面对的被积函数,直接寻求其原函数往往很困难,甚至是不可能的.与此不同,机械求积

方法将积分求值问题归结为提供若干节点上的函数值,从而使问题获得了简化.

2. 为要设计机械求积公式,需要提供求积节点及其相应的求积系数(权系数).求积公式的设计是一个确定这些参数的代数问题.

机械求积是一类近似方法.为要保证这类方法行之有效,要求它具有足够的精度.本章侧重于从代数精度的角度审视求积公式,令它对于次数尽可能高的多项式能准确成立,从而将求积公式的设计归结为求解代数方程组.

3. 求积公式的设计要区分两种情况.如果求积节点事先给定,譬如取求积区间上的等分点作为求积节点,则所归结出的方程组是线性的,这时处理过程比较简单.据此可设计出一大类 Newton - Cotes 公式.在实用上,有价值的 Newton - Cotes 公式主要有梯形公式、Simpson 公式和 Cotes 公式.

如果求积节点不加限定,而是灵活地选取,则可能进一步提高求积公式的精度.这类求积公式称为 Gauss 公式.

表面上看,设计 Gauss 公式要面对非线性方程组,从而存在实质性的困难.其实,“好”的数学对象必然是“美”的,而对称美则是数学美的一个重要标志.可以看到,基于对称性原则很容易化解 Gauss 公式设计过程中的难点.对称性威力巨大!

值得指出的是,Gauss 型求积公式通常用于计算无穷积分、奇异积分等特殊类型的积分.

4. 为改善求积方法的精度,亦可仿照插值方法采用分段技术,即事先将求积区间划分为若干等分,然后在每个子段(子段长度称步长)上套用低阶求积公式计算积分值.这类求积方法称作是复化的.

步长的合理选取是运用复化求积方法的关键.步长太大精度难以保证,步长太小则会导致计算量的浪费,然而事先给出一个合适的步长往往是困难的.

通常采用变步长的计算方案,即在等分数逐步倍增——相应地步长逐次减半的二分过程中计算积分值.每做一步,检查一下二分前后计算结果的偏差,直到满足精度要求时终止计算.

5. 在二分过程中考察几种低阶求积公式的联系,不难发现,二分前后两个梯形值适当组合可以获得 Simpson 值,而二分前后两个 Simpson 值适当组合又可进一步获得 Cotes 值.如果将二分前后两个 Cotes 值再适当组合即可获得更高精度的 Romberg 值.这样,先在二分过程中逐步计算出梯形值序列,然后再将它逐步加工成 Simpson 值序列、Cotes 值序列与 Romberg 值序列.这就是著名的 Romberg 算法.

Romberg 算法在几乎不增加计算量的前提下显著地提高了计算结果的精度,其加速效果是奇妙的.它是优秀数值算法的一个范例.

6. 数值求积的 Romberg 加速算法是 20 世纪中(1955 年)才提出来的.令人

感到不可思议的是，早在公元3世纪（公元263年以前），魏晋大数学家刘徽在圆周率的割圆计算中，就已经运用了今日被称为松弛技术的加速技术．这是一项超越时代的辉煌成就．这是中华先贤的大智慧．

题解2.1 求积公式的设计

提要 本讲突出设计求积公式的代数精度方法．所设计的求积公式应具有“尽可能高”的代数精度，并要求指明所构造出的求积公式实际具有的代数精度．

值得强调指出的是，在设计求积公式时要充分考虑对称性．利用对称性可以减少待定参数的数目，从而使所归结出的代数方程组较为容易求解．某些具有对称结构的求积公式，其代数精度获得了“额外”的好处．

题1 试设计求积公式

$$\int_0^1 f(x)\mathrm{d}x \approx A_0 f\left(\frac{1}{4}\right)+A_1 f\left(\frac{3}{4}\right)$$

解 令原式对于 $f=1, f=x$ 准确成立，可列出方程组

$$\begin{cases} A_0+A_1=1 \\ \frac{1}{4}A_0+\frac{3}{4}A_1=\frac{1}{2} \end{cases}$$

解得 $A_0=A_1=\frac{1}{2}$．这样设计出的求积公式是

$$\int_0^1 f(x)\mathrm{d}x \approx \frac{1}{2}\left[f\left(\frac{1}{4}\right)+f\left(\frac{3}{4}\right)\right]$$

令 $f=x^2$，上式

$$左端=\int_0^1 x^2\mathrm{d}x=\frac{1}{3}$$

$$右端=\frac{1}{2}\left(\frac{1}{16}+\frac{9}{16}\right)$$

左右两端不相等，故所设计出的求积公式仅有一阶精度．

题2 试设计求积公式

$$\int_0^1 f(x)\mathrm{d}x \approx A_0 f\left(\frac{1}{4}\right)+A_1 f\left(\frac{1}{2}\right)+A_2 f\left(\frac{3}{4}\right)$$

解 令原式对于 $f=1,x,x^2$ 准确成立，可列出方程组

$$\begin{cases} A_0+A_1+A_2=1 \\ \frac{1}{4}A_0+\frac{1}{2}A_1+\frac{3}{4}A_2=\frac{1}{2} \\ \frac{1}{16}A_0+\frac{1}{4}A_1+\frac{9}{16}A_2=\frac{1}{3} \end{cases}$$

考虑到对称性，令 $A_2=A_0$，则方程组化简为

$$\begin{cases} A_0+\dfrac{1}{2}A_1=\dfrac{1}{2} \\ \dfrac{5}{8}A_0+\dfrac{1}{4}A_1=\dfrac{1}{3} \end{cases}$$

解之得 $A_0=A_2=\dfrac{2}{3}, A_1=-\dfrac{1}{3}$. 易知所构造出的求积公式

$$\int_0^1 f(x)\mathrm{d}x \approx \frac{2}{3}f\left(\frac{1}{4}\right)-\frac{1}{3}f\left(\frac{1}{2}\right)+\frac{2}{3}f\left(\frac{3}{4}\right)$$

对于 $f=x^3$ 准确成立，而对于 $f=x^4$ 不准确，故它有三阶精度.

题 3 试设计求积公式

$$\int_0^1 f(x)\mathrm{d}x \approx A_0 f(0)+A_1 f(1)+B_0 f'(0)$$

解 令原式对 $f=1, x, x^2$ 准确成立，可列出方程组

$$\begin{cases} A_0+A_1=1 \\ A_1+B_0=\dfrac{1}{2} \\ A_1=\dfrac{1}{3} \end{cases}$$

解之有

$$A_0=\frac{2}{3},\ A_1=\frac{1}{3},\ B_0=\frac{1}{6}$$

易知这样设计出的求积公式

$$\int_0^1 f(x)\mathrm{d}x \approx \frac{2}{3}f(0)+\frac{1}{3}f(1)+\frac{1}{6}f'(0)$$

对于 $f=x^3$ 不准确，故它仅有二阶精度.

题 4 试设计求积公式

$$\int_0^h f(x)\mathrm{d}x \approx h[a_0 f(0)+a_1 f(1)]+h^2[b_0 f'(0)+b_1 f'(1)]$$

解 不妨设 $h=1$（否则作变换 $x=ht$），而考察下列求积公式

$$\int_0^1 f(x)\mathrm{d}x \approx a_0 f(0)+a_1 f(1)+b_0 f'(0)+b_1 f'(1)$$

令对于 $f=1, x, x^2, x^3$ 准确成立，可列出方程组

$$\begin{cases} a_0 + a_1 = 1 \\ a_1 + b_0 + b_1 = \dfrac{1}{2} \\ a_1 + 2b_1 = \dfrac{1}{3} \\ a_1 + 3b_1 = \dfrac{1}{4} \end{cases}$$

解之得 $a_0 = a_1 = \frac{1}{2}, b_0 = -b_1 = \frac{1}{12}$,易知这样设计出的求积公式

$$\int_0^h f(x)\mathrm{d}x \approx \frac{h}{2}[f(0) + f(1)] + \frac{h^2}{12}[f'(0) - f'(1)]$$

对于 $f = x^4$不准确,故它有三阶精度.

题 5 试设计求积公式

$$\int_{-h}^{h} f(x)\mathrm{d}x \approx A_0 f(-h) + A_1 f(x_1)$$

解 所要构造的求积公式含有未知的求积节点 x_1,令它对于 $f = 1, x, x^2$ 准确成立,可列出方程组

$$\begin{cases} A_0 + A_1 = 2h \\ -hA_0 + x_1 A_1 = 0 \\ h^2 A_0 + x_1^2 A_1 = \dfrac{2}{3}h^3 \end{cases}$$

利用前两个方程从最末一个方程中消去 A_1 与 x_1,使之化归为仅含一个变元 A_0 的方程,从而定出 $A_0 = \frac{h}{2}$. 再代入求得 $A_1 = \frac{3}{2}h, x_1 = \frac{h}{3}$. 于是所设计的求积公式是

$$\int_{-h}^{h} f(x)\mathrm{d}x \approx \frac{h}{2} f(-h) + \frac{3}{2} h f\left(\frac{h}{3}\right)$$

易知这一求积公式当 $f = x^3$时不准确,故它仅有二阶精度.

题 6 试设计求积公式

$$\int_{-1}^{1} f(x)\mathrm{d}x \approx A[f(x_0) + f(x_1) + f(x_2)], \quad x_0 < x_1 < x_2$$

解 考虑到求积公式内在的对称性,令 $x_0 = -x_2, x_1 = 0$,则原式化为

$$\int_{-1}^{1} f(x)\mathrm{d}x \approx A[f(-x_2) + f(0) + f(x_2)]$$

这样设计出的求积公式对于奇函数 $f = x, x^3, x^5$ 等均准确成立,再令它对于 $f = 1$准确成立,列出方程

$$3A = 2$$

因而 $A=2/3$. 进一步令它对于 $f=x^2$ 准确成立,可列出方程

$$\frac{2}{3}(2x_2^2)=\frac{2}{3}$$

由此得知 $x_2=-x_0=\frac{1}{\sqrt{2}}$. 从而构造出求积公式

$$\int_{-1}^{1} f(x)\mathrm{d}x \approx \frac{2}{3}\left[f\left(-\frac{1}{\sqrt{2}}\right)+f(0)+f\left(\frac{1}{\sqrt{2}}\right)\right]$$

易知它对于 $f=x^4$ 不准确,故这一公式有三阶精度.

题解 2.2　Gauss 求积公式

提要　Gauss 公式是一类高精度的求积公式. 这类求积公式还具备其他一系列优点,譬如数值稳定性好,适合于处理某些奇异积分等.

不过,Gauss 公式的设计有实质性的困难,为了同时处理求积系数与求积节点,用代数精度方法归结出的代数方程组是非线性的. 前文 2.3 节揭露出一个令人感兴趣的事实:利用 Gauss 公式的对称结构可以使处理过程大大地简化.

题 1　证明下列求积公式有 5 阶精度:

$$\int_{1}^{3} f(x)\mathrm{d}x \approx \frac{5}{9}f\left(2-\sqrt{\frac{3}{5}}\right)+\frac{8}{9}f(2)+\frac{5}{9}f\left(2+\sqrt{\frac{3}{5}}\right)$$

证　这个公式含有 3 个节点

$$x_0=2-\sqrt{\frac{3}{5}},\quad x_1=2,\quad x_2=2+\sqrt{\frac{3}{5}}$$

为使它具有 5 阶精度,它必须是 Gauss 公式. 容易看出,若引进变换 $x=t+2$,则可将它变到三点 Gauss 公式

$$\int_{-1}^{1} f(t)\mathrm{d}t \approx \frac{5}{9}f\left(-\sqrt{\frac{3}{5}}\right)+\frac{8}{9}f(0)+\frac{5}{9}f\left(\sqrt{\frac{3}{5}}\right)$$

题 2　试设计求积公式

$$\int_{-2}^{2} f(x)\mathrm{d}x \approx Af(-a)+Bf(0)+Cf(a)$$

解　注意到它的对称结构,作变换 $x=2t$,原式变成

$$\int_{-1}^{1} f(2t)\mathrm{d}t \approx \frac{A}{2}f\left(-\frac{a}{2}\right)+\frac{B}{2}f(0)+\frac{C}{2}f\left(\frac{a}{2}\right)$$

与三点 Gauss 公式

$$\int_{-1}^{1} f(x)\mathrm{d}x \approx \frac{5}{9}f\left(-\sqrt{\frac{3}{5}}\right)+\frac{8}{9}f(0)+\frac{5}{9}f\left(\sqrt{\frac{3}{5}}\right)$$

比较知

$$A=C=\frac{10}{9},\quad B=\frac{16}{9},\quad a=2\sqrt{\frac{3}{5}}$$

题 3 构造求积公式

$$\int_{-1}^{1}(1+x^2)f(x)\mathrm{d}x\approx A_0f(x_0)+A_1f(x_1)$$

使具有三阶精度.

解 令对于 $f=1,x,x^2,x^3$准确成立,可列出方程组

$$\begin{cases}A_0+A_1=\dfrac{8}{3}\\ A_0x_0+A_1x_1=0\\ A_0x_0^2+A_1x_1^2=\dfrac{16}{15}\\ A_0x_0^3+A_1x_1^3=0\end{cases}$$

考虑到对称性,令 $A_0=A_1,x_0=-x_1$,则有

$$A_0=A_1=\frac{4}{3}$$

$$x_0^2=x_1^2=\frac{2}{5}$$

故有 $x_0=-x_1=-\sqrt{\dfrac{2}{5}}$.故所要设计的求积公式是

$$\int_{-1}^{1}(1+x^2)f(x)\mathrm{d}x\approx\frac{4}{3}\left[f\left(-\sqrt{\frac{2}{5}}\right)+f\left(\sqrt{\frac{2}{5}}\right)\right]$$

题 4 验证 Gauss 型求积公式

$$\int_0^{\infty}\mathrm{e}^{-x}f(x)\mathrm{d}x\approx A_0f(x_0)+A_1f(x_1)$$

的系数及节点分别为

$$A_0=\frac{\sqrt{2}+1}{2\sqrt{2}},\quad A_1=\frac{\sqrt{2}-1}{2\sqrt{2}}$$

$$x_0=2-\sqrt{2},\quad x_1=2+\sqrt{2}$$

证 令原式对于 $f=1,x,x^2,x^3$准确成立,注意到对正整数 n 有

$$\int_0^{\infty}x^n\mathrm{e}^{-x}\mathrm{d}x=n!$$

可列出方程组

$$\begin{cases}A_0+A_1=1\\ A_0x_0+A_1x_1=1\\ A_0x_0^2+A_1x_1^2=2\\ A_0x_0^3+A_1x_1^3=6\end{cases}$$

直接验证知所给节点 x_0, x_1 与系数 A_0, A_1 确实满足这个方程组.

习 题 二

1. 直接验证梯形公式与中矩形公式具有一阶代数精度,而 Simpson 公式则有三阶代数精度.

2. 试判定下列求积公式的代数精度:

$$\int_0^1 f(x)\mathrm{d}x \approx \frac{3}{4} f\left(\frac{1}{3}\right) + \frac{1}{4} f(1)$$

3. 确定下列求积公式中的待定参数,使其代数精度尽可能地高,并指明求积公式所具有的代数精度:

(1) $\int_{-h}^{h} f(x)\mathrm{d}x \approx A_0 f(-h) + A_1 f(0) + A_2 f(h)$

(2) $\int_0^1 f(x)\mathrm{d}x \approx \frac{1}{4} f(0) + A_0 f(x_0)$

4. 下列求积公式称作 Simpson3/8 公式:

$$\int_0^3 f(x)\mathrm{d}x \approx \frac{3}{8}[f(0) + 3f(1) + 3f(2) + f(3)]$$

试判断这一求积公式的代数精度.

5. 试设计求积公式

$$\int_0^1 f(x)\mathrm{d}x \approx A_0 f(0) + A_1 f(x_1) + A_2 f(1)$$

6. 验证求积公式

$$\int_1^3 f(x)\mathrm{d}x \approx \frac{5}{9} f\left(2 - \sqrt{\frac{3}{5}}\right) + \frac{8}{9} f(2) + \frac{5}{9} f\left(2 + \sqrt{\frac{3}{5}}\right)$$

是三点 Gauss 公式.

7. 试将二分前后的中矩形公式

$$M_1 = (b-a) f\left(\frac{a+b}{2}\right)$$

$$M_2 = \frac{b-a}{2}\left[f\left(\frac{3a+b}{4}\right) + f\left(\frac{a+3b}{4}\right)\right]$$

加工成松弛公式

$$\int_a^b f(x)\mathrm{d}x \approx M_2 + \omega(M_2 - M_1)$$

使其代数精度尽可能高.

8. 直接验证下列数值微分方法的代数精度:

(1) 前差公式 $f'(a) \approx \frac{f(a+h) - f(a)}{h}$

(2) 后差公式 $f'(a) \approx \frac{f(a) - f(a-h)}{h}$

(3) 中差公式 $f'(a) \approx \frac{f(a+h) - f(a-h)}{2h}$

9. 证明下列数值微分公式具有 4 阶代数精度：

$$f'(x_0)\approx\frac{1}{12h}[f(x_0-2h)-8f(x_0-h)+8f(x_0+h)-f(x_0+2h)]$$

第三章 常微分方程的差分法

科学计算中常常需要求解常微分方程的定解问题.这类问题的最简形式,是本章将要着重考察的一阶方程的初值问题.

$$\begin{cases} y' = f(x, y) \\ y(x_0) = y_0 \end{cases} \tag{1}$$

这里假定右函数 $f(x, y)$ 适当光滑,譬如关于 y 满足 Lipschitz 条件,以保证上述初值问题的解 $y(x)$ 存在且唯一.

虽然求解常微分方程有各种各样的解析方法,但解析方法只能用来求解一些特殊类型的方程.求解从实际问题当中归结出来的微分方程主要靠数值解法.

差分方法是一类重要的数值解法.这类方法回避解 $y(x)$ 的函数表达式,而是寻求它在一系列离散节点

$$x_0 < x_1 < x_2 < \cdots < x_n < \cdots$$

上的近似值 $y_0, y_1, y_2, \cdots, y_n, \cdots$.相邻两个节点的间距 $h = x_{i+1} - x_i$ 称**步长**.本章将假定步长 h 为定数.

差分方法是一类离散化方法,这种方法将寻求解 $y(x)$ 的分析问题转化为计算离散值 y_n 的代数问题,然而随之带来的困难是,由于数据量 $\{y_n\}$ 通常很大,差分方法所归结出的通常是个大规模的代数方程组.

初值问题的各种差分方法有个基本特点,它们都采取"步进式",即求解过程顺着节点排列的次序一步一步地向前推进.描述这类算法,只要给出从已知信息 $y_n, y_{n-1}, y_{n-2}, \cdots$ 计算 y_{n+1} 的递推公式.这类计算公式称为**差分格式**.

差分格式中仅含一个未知参数 y_{n+1},或者说,它是仅含一个变元 y_{n+1} 的代数方程,这就大大地缩减了计算问题的规模.

总之,差分方法的设计思想是,将寻求微分方程的解 $y(x)$ 的分析问题化归为计算离散值 $\{y_n\}$ 的代数问题,而"步进式"则进一步将计算模型化归为仅含一个变元 y_{n+1} 的代数方程——所谓差分格式,这就达到了化繁为简的目的.

3.1 Euler 方法

方程(1)中含有导数项 $y'(x)$,这是微分方程的本质特征,也正是它难以求

解的症结所在.导数是无穷极限过程的结果,而计算过程则总是有限的.因此,数值解法的第一步就是消除式(1)中的导数项 y',这项手续称**离散化**.由于差商是微分的近似运算,实现离散化的一种直截了当的途径是用差商替代导数.

3.1.1 Euler 格式

设在区间$[x_n, x_{n+1}]$的左端点 x_n 列出方程(1):

$$y'(x_n) = f(x_n, y(x_n))$$

并用差商$\frac{y(x_{n+1}) - y(x_n)}{h}$替代其中的导数项$y'(x_n)$,则有

$$y(x_{n+1}) \approx y(x_n) + hf(x_n, y(x_n)) \tag{2}$$

若用 $y(x_n)$的近似值 y_n 代入上式右端,并记所得结果为 y_{n+1},这样设计出的计算公式

$$y_{n+1} = y_n + hf(x_n, y_n), \quad n = 0,1,2,\cdots \tag{3}$$

就是著名的 **Euler 格式**.若初值 y_0 已知,则依格式(3)可逐步算出数值解 $y_1, y_2, \cdots$.

例 1 求解初值问题

$$\begin{cases} y' = y - \dfrac{2x}{y}, & x \in [0,1] \\ y(0) = 1 \end{cases} \tag{4}$$

解 为便于进行比较,本章将用多种差分方法求解上述初值问题.这里先用 Euler 方法,求解方程(4)的 Euler 格式具有形式

$$y_{n+1} = y_n + h\left(y_n - \frac{2x_n}{y_n}\right)$$

取步长 $h = 0.1$,Euler 格式的计算结果见表 3.1.

表 3.1

x_n	y_n	$y(x_n)$	x_n	y_n	$y(x_n)$
0.1	1.100 0	1.095 4	0.6	1.509 0	1.483 2
0.2	1.191 8	1.183 2	0.7	1.580 3	1.549 2
0.3	1.277 4	1.264 9	0.8	1.649 8	1.612 5
0.4	1.358 2	1.341 6	0.9	1.717 8	1.673 3
0.5	1.435 1	1.414 2	1.0	1.784 8	1.732 1

初值问题(4)有解析解 $y = \sqrt{1+2x}$,这里将解的准确值 $y(x_n)$同近似值 y_n 一起列在表 3.1 中,比较两者可以看出 Euler 格式的精度很低.

再从图形上看，假设节点 $P_n(x_n, y_n)$ 位于积分曲线 $y=y(x)$ 上，则按 Euler 格式定出的节点 $P_{n+1}(x_{n+1}, y_{n+1})$ 落在积分曲线 $y=y(x)$ 的切线上(图 3.1)，从这个角度也可以看出 Euler 格式是很粗糙的.

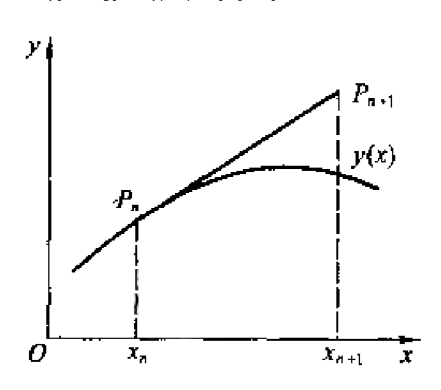


图 3.1 Euler 格式的几何解释

3.1.2 隐式 Euler 格式

再在区间 $[x_n, x_{n+1}]$ 的右端点 x_{n+1} 列出方程(1)：

$$y'(x_{n+1}) = f(x_{n+1}, y(x_{n+1}))$$

并改用向后差商 $\frac{y(x_{n+1}) - y(x_n)}{h}$ 替代方程中的导数项 $y'(x_{n+1})$，再离散化，即可导出下列**隐式 Euler 格式**

$$y_{n+1} = y_n + hf(x_{n+1}, y_{n+1}) \tag{5}$$

这一格式与 Euler 格式(3)有着本质的区别：Euler 格式(3)是关于 y_{n+1} 的一个直接的计算公式，这类格式称作是**显式的**；而格式(5)的右端含有未知的 y_{n+1}，它实际上是个关于 y_{n+1} 的函数方程(关于函数方程的解法将在下一章介绍). 这类格式称作是**隐式的**. 隐式格式的计算远比显式格式困难.

由于数值微分的向前差商公式与向后差商公式具有同等精度，可以预料，隐式 Euler 格式(5)与显式 Euler 格式(3)的精度相当.

3.1.3 Euler 两步格式

为了改善精度，可以改用中心差商 $\frac{1}{2h}[y(x_{n+1}) - y(x_{n-1})]$ 替代方程 $y'(x_n) = f(x_n, y(x_n))$ 中的导数项，再离散化，即可导出下列格式

$$y_{n+1} = y_{n-1} + 2hf(x_n, y_n) \tag{6}$$

无论是显式 Euler 格式(3)还是隐式 Euler 格式(5)，它们都是**单步法**，其特点是计算 y_{n+1} 时只用到前一步的信息 y_n；然而格式(6)除了 y_n 以外，还显含更

前一步的信息 y_{n-1}，即调用了前面两步的信息，**Euler 两步格式**因此而得名.

Euler 两步格式(6)虽然比 Euler 格式或隐式 Euler 格式具有更高的精度，但它是一种两步法. 两步法不能自行启动，实际使用时除初值 y_0 外还需要借助于某种一步法再提供一个**开始值** y_1，这就增加了计算程序的复杂性.

3.1.4 梯形格式

设将方程 $y'=f(x,y)$ 的两端从 x_n 到 x_{n+1} 求积分，即得

$$y(x_{n+1})=y(x_n)+\int_{x_n}^{x_{n+1}}f(x,y(x))\mathrm{d}x \tag{7}$$

显然，为要通过这个积分关系式获得 $y(x_{n+1})$ 的近似值，只要近似地算出其中的积分项 $\int_{x_n}^{x_{n+1}}f(x,y(x))\mathrm{d}x$，而选用不同的计算方法计算这个积分项，就会得到不同的差分格式.

例如，设用矩形方法计算积分项

$$\int_{x_n}^{x_{n+1}}f(x,y(x))\mathrm{d}x\approx hf(x_n,y(x_n))$$

代入式(7)，有

$$y(x_{n+1})\approx y(x_n)+hf(x_n,y(x_n))$$

据此离散化又可导出 Euler 格式(3). 由于数值积分的矩形方法精度很低，Euler 格式当然很粗糙.

为了提高精度，改用梯形方法计算积分项

$$\int_{x_n}^{x_{n+1}}f(x,y(x))\mathrm{d}x\approx\frac{h}{2}[f(x_n,y(x_n))+f(x_{n+1},y(x_{n+1}))]$$

再代入式(7)，有

$$y(x_{n+1})\approx y(x_n)+\frac{h}{2}[f(x_n,y(x_n))+f(x_{n+1},y(x_{n+1}))]$$

设将式中的 $y(x_n)$，$y(x_{n+1})$ 分别用 y_n，y_{n+1} 替代，作为离散化的结果导出下列计算格式

$$y_{n+1}=y_n+\frac{h}{2}[f(x_n,y_n)+f(x_{n+1},y_{n+1})] \tag{8}$$

与梯形求积公式相呼应的这一差分格式称作**梯形格式**.

容易看出，梯形格式(8)实际上是显式 Euler 格式(3)与隐式 Euler 格式(5)的算术平均.

3.1.5 改进的 Euler 格式

Euler 格式(3)是一种显式算法，其计算量小，但精度很低；梯形格式(8)虽提

高了精度，但它是一种隐式算法，需要借助于迭代过程求解，计算量大.

可以综合使用这两种方法，先用 Euler 格式求得一个初步的近似值，记为 $\bar{y}_{n+1}$ 称之为**预报值**；预报值的精度不高，用它替代式(8)右端的 y_{n+1} 再直接计算，得到**校正值** y_{n+1}. 这样建立的**预报校正系统**

$$\text{预报}\quad \bar{y}_{n+1}=y_n+hf(x_n,y_n)$$

$$\text{校正}\quad y_{n+1}=y_n+\frac{h}{2}[f(x_n,y_n)+f(x_{n+1},\bar{y}_{n+1})] \tag{9}$$

称作**改进的 Euler 格式**. 这是一种显式格式，它可表示为如下嵌套形式

$$y_{n+1}=y_n+\frac{h}{2}[f(x_n,y_n)+f(x_{n+1},y_n+hf(x_n,y_n))]$$

或表示为下列平均化形式

$$\begin{cases} y_p=y_n+hf(x_n,y_n) \\ y_c=y_n+hf(x_{n+1},y_p) \\ y_{n+1}=\dfrac{1}{2}(y_p+y_c) \end{cases} \tag{10}$$

图 3.2 描述了改进的 Euler 方法，其中 h 为步长，N 为步数，x_0,y_0 为"老值"，即前一步的近似解，x_1,y_1 为"新值"，即该步计算结果.

例 2 用改进的 Euler 格式求解初值问题(4).

解 求解初值问题的改进的 Euler 格式(10)具有形式

$$\begin{cases} y_p=y_n+h\left(y_n-\dfrac{2x_n}{y_n}\right) \\ y_c=y_n+h\left(y_p-\dfrac{2x_{n+1}}{y_p}\right) \\ y_{n+1}=\dfrac{1}{2}(y_p+y_c) \end{cases}$$

仍取 $h=0.1$，计算结果见表 3.2. 同例 1 所列的 Euler 格式的计算结果(表 3.1)比较，改进的 Euler 格式明显地改善了精度.

表 3.2

x_n	y_n	$y(x_n)$	x_n	y_n	$y(x_n)$
0.1	1.095 9	1.095 4	0.6	1.486 0	1.483 2
0.2	1.184 1	1.183 2	0.7	1.552 5	1.549 2
0.3	1.266 2	1.264 9	0.8	1.616 5	1.612 5
0.4	1.343 4	1.341 6	0.9	1.678 2	1.673 3
0.5	1.416 4	1.414 2	1.0	1.737 9	1.732 1

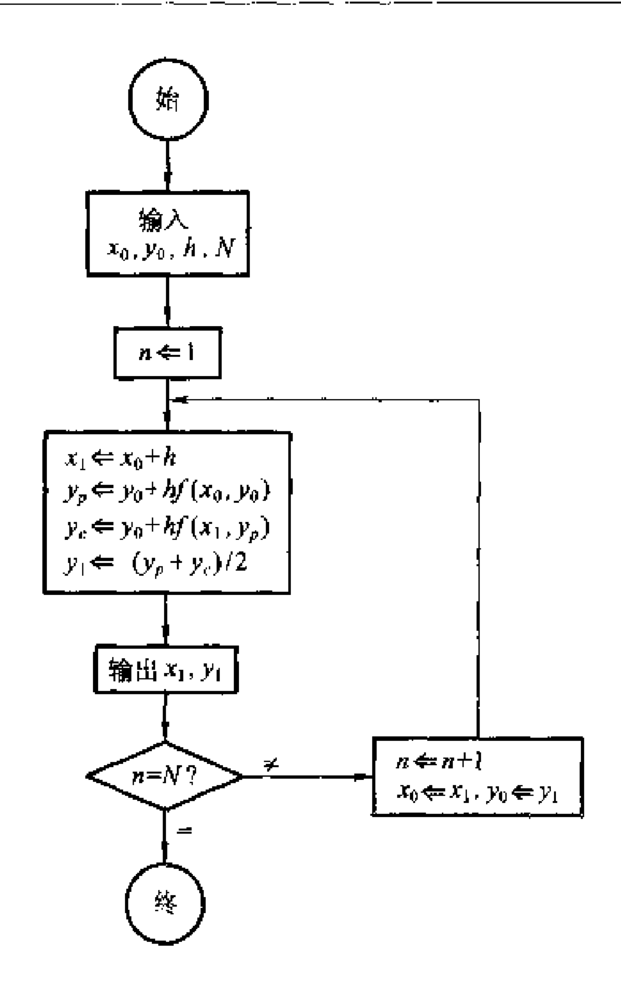


图 3.2 改进的 Euler 方法的计算流程

3.1.6 Euler 方法的分类

前述 Euler 方法分**显式格式与隐式格式**两大类. Euler 格式与 Euler 两步格式是显式的,而隐式 Euler 格式与梯形格式则是隐式的. 显式格式与隐式格式各有利弊:显式格式的计算量小,但稳定性较差;与此相反,隐式格式的稳定性好,但需要通过迭代法求解,计算量比较大.

相反恰恰相辅相成. 实际应用时往往综合应用显式与隐式两种格式,即先用显式格式求得某个预报值,然后再用隐式格式迭代一次得出较高精度的校正值. 改进的 Euler 格式即是这种**预报校正系统**.

Euler 方法又有**一步法**与**多步法**之分. 在前述几种 Euler 格式中,Euler 两步格式以外的几种格式都是一步法,顾名思义,Euler 两步格式则是两步法. 比较一步法与多步法,前者计算量大,而后者由于使用了前面多步的老信息,不需要增加计算量即可获得高精度. 不过多步法也有缺陷:它不能自行启动,必须依赖某种一步法为它提供所需的开始值,这就导致程序结构的复杂性.

后文的 3.2 节、3.3 节分别考察一步法的 Runge - Kutta 方法与多步法的

Adams 方法,它们分别是 Euler 方法的延伸与拓广.

3.1.7 Euler 方法的精度分析

类似于前面两章的处理方法,本章依然运用代数精度来判定差分格式的精度.

定义 1 称某个差分格式具有 m **阶精度**,如果其对应的近似关系式对于次数$\leqslant m$ 的多项式均能准确成立,而对 $m+1$ 次式不能准确成立.

譬如,考察 Euler 格式(3)

$$y_{n+1}=y_n+hf(x_n,y_n)$$

其对应的近似关系式为

$$y(x_{n+1})\approx y(x_n)+hy'(x_n)$$

检验它所具有的代数精度,当 $y=1$ 时,

$$左端=右端=1$$

当 $y=x$ 时

$$左端=右端=x_n+h$$

而当 $y=x^2$ 时

$$左端=x_{n+1}^2=(x_n+h)^2$$

$$右端=x_n^2+2hx_n$$

这时左端$\neq$右端,可见 Euler 格式仅有一阶精度.

类似地不难验证隐式 Euler 格式同样仅有一阶精度.

再考察梯形格式(8)

$$y_{n+1}=y_n+\frac{h}{2}[f(x_n,y_n)+f(x_{n+1},y_{n+1})]$$

其对应的近似关系式为

$$y(x_{n+1})\approx y(x_n)+\frac{h}{2}[y'(x_n)+y'(x_{n+1})] \tag{11}$$

值得指出的是,为简化处理手续,可引进变换 $x=x_n+th$,而不妨令节点 $x_n=0$,步长 $h=1$,从而将近似关系式化简.这时,梯形格式的近似关系式(11)化简为

$$y(1)\approx y(0)+\frac{1}{2}[y'(0)+y'(1)]$$

易知它对 $y=1,x,x^2$ 均准确成立,而当 $y=x^3$ 时左端$=1$,右端$=\frac{3}{2}$,因而梯形格式具有二阶精度.

比较几种 Euler 方法.Euler 格式是显式计算,计算量小,结构简单,但精度低;梯形格式改善了精度,但它是隐式的,求解困难.相比之下,改进的 Euler 格

式无论计算量还是精度都是可取的．自然会问，能否推广改进的 Euler 格式以进一步提高差分格式的精度呢？

3.2 Runge－Kutta 方法

3.2.1 Runge－Kutta 方法的设计思想

考察差商 $\frac{y(x_{n+1})-y(x_n)}{h}$，根据微分中值定理，存在点 ξ，$x_n<\xi<x_{n+1}$，使得

$$\frac{y(x_{n+1})-y(x_n)}{h}=y'(\xi)$$

从而利用所给方程 $y'=f$ 得

$$y(x_{n+1})=y(x_n)+hf(\xi,y(\xi)) \tag{12}$$

其中的 $K^*=f(\xi,y(\xi))$ 称作区间 $[x_n,x_{n+1}]$ 上的**平均斜率**．这样，只要对平均斜率 K^* 提供一种算法，由式(12)便相应地导出一种计算格式．

按照这种观点考察 Euler 格式(3)，它简单地取点 x_n 的斜率值 $K_1=f(x_n,y_n)$ 作为平均斜率 K^*，精度自然很低．

再考察改进的 Euler 格式(9)，它可改写成下列平均化形式：

$$\begin{cases} y_{n+1}=y_n+\frac{h}{2}(K_1+K_2) \\ K_1=f(x_n,y_n) \\ K_2=f(x_{n+1},y_n+hK_1) \end{cases} \tag{13}$$

因之可以理解为：它用 x_n 与 x_{n+1} 两个点的斜率值 K_1 和 K_2 取算术平均作为平均斜率 K^*，而 x_{n+1} 处的斜率值 K_2 则利用已知信息 y_n 通过 Euler 格式来预报．

这个处理过程启示我们，如果设法在 $[x_n,x_{n+1}]$ 内多预报几个点的斜率值，然后将它们加权平均作为平均斜率 K^*，则有可能构造出更高精度的计算格式，这就是 Runge－Kutta 方法的设计思想．

3.2.2 中点格式

再考察 Euler 两步格式(6)：

$$\begin{cases} y_{n+1}=y_{n-1}+2hy'_n \\ y'_n=f(x_n,y_n) \end{cases}$$

这一格式用区间 $[x_{n-1},x_{n+1}]$ 中点 x_n 的斜率值 y'_n 作为该区间上的平均斜率．不

难验证它有二阶精度.因之,如果改用区间$[x_n, x_{n+1}]$中点$x_{n+\frac{1}{2}}$的斜率值$y'_{n+\frac{1}{2}}$作为该区间上的平均斜率,则所设计出的差分格式

$$y_{n+1} = y_n + h y'_{n+\frac{1}{2}} \tag{14}$$

亦应有二阶精度.问题在于该如何生成$y'_{n+\frac{1}{2}}$?

设y_n为已知,则可用 Euler 格式预报$y_{n+\frac{1}{2}}$:

$$y_{n+\frac{1}{2}} = y_n + \frac{h}{2} y'_n$$

从而有

$$y'_{n+\frac{1}{2}} = f(x_{n+\frac{1}{2}}, y_{n+\frac{1}{2}})$$

这样设计出的格式

$$\begin{cases} y_{n+1} = y_n + hK_2 \\ K_1 = f(x_n, y_n) \\ K_2 = f(x_{n+\frac{1}{2}}, y_n + \frac{h}{2} K_1) \end{cases} \tag{15}$$

称作**变形的 Euler 格式**,或称**中点格式**.

表面上看,中点格式$y_{n+1} = y_n + hK_2$中仅含一个斜率值K_2,然而K_2是通过K_1计算出来的,因此它每做一步仍然需要两次计算函数f的值,工作量和改进的 Euler 格式(13)相同.

例 3 用中点格式(15)求解初值问题(4).

解 仍取$h=0.1$,计算结果见表 3.3.比较例 2 可以看到,中点格式与改进的 Euler 格式精度相当.

表 3.3

x_n	y_n	$y(x_n)$	x_n	y_n	$y(x_n)$
0.1	1.095 5	1.095 4	0.6	1.483 7	1.483 2
0.2	1.183 3	1.183 2	0.7	1.549 8	1.549 2
0.3	1.265 1	1.264 9	0.8	1.613 2	1.612 5
0.4	1.341 9	1.341 6	0.9	1.674 2	1.673 3
0.5	1.414 6	1.414 2	1.0	1.733 1	1.732 1

3.2.3 二阶 Runge－Kutta 方法

推广改进的 Euler 格式(13)与中点格式(15).对于区间$[x_n, x_{n+1}]$内任意给定的一点x_{n+p}:

$$x_{n+p}=x_n+ph,\quad 0<p\leqslant 1$$

设用 x_n 和 x_{n+p} 两个点的斜率值 K_1 和 K_2 加权平均得到平均斜率 K^*，即令

$$y_{n+1}=y_n+h[(1-\lambda)K_1+\lambda K_2] \tag{16}$$

式中 λ 为待定参数. 同改进的 Euler 格式(13)以及中点格式(15)一样，这里仍取 $K_1=f(x_n,y_n)$，问题在于该怎样预报 x_{n+p} 处的斜率值 K_2?

仿照格式(13)与(15)，先用 Euler 格式提供 $y(x_{n+p})$ 的预报值 y_{n+p}:

$$y_{n+p}=y_n+phK_1$$

然后用 y_{n+p} 通过计算 f 产生斜率值

$$K_2=f(x_{n+p},y_{n+p})$$

这样设计出的计算格式具有形式

$$\begin{cases}y_{n+1}=y_n+h[(1-\lambda)K_1+\lambda K_2]\\ K_1=f(x_n,y_n)\\ K_2=f(x_{n+p},y_n+phK_1)\end{cases} \tag{17}$$

问题在于，如何选取参数 λ 的值，使得格式(17)具有较高的精度.

为此考察格式(16)对应的近似关系式，注意到其中的 K_1,K_2 分别代表点 x_n,x_{n+p} 处的斜率值，有

$$y(x_{n+p})\approx y(x_n)+h[(1-\lambda)y'(x_n)+\lambda y'(x_{n+p})]$$

容易看出，不管 λ 如何选取，上式均有一阶精度. 可以适当选取参数 λ，使上式具有二阶精度. 为简化处理，仍令 $x_n=0,h=1$，这时上述近似关系式化简为

$$y(p)\approx y(0)+(1-\lambda)y'(0)+\lambda y'(p)$$

令它对于 $y=x^2$ 准确成立，得知

$$\lambda\cdot p=\frac{1}{2} \tag{18}$$

满足这项条件的一族格式(17)统称二阶 **Runge－Kutta 格式**.

二阶 Runge－Kutta 格式有两个重要的特例. 当 $p=1,\lambda=\frac{1}{2}$ 时格式(17)是改进的 Euler 格式(13)；而如果取 $p=\frac{1}{2},\lambda=1$，这时二阶 Runge－Kutta 格式是中点格式(15).

3.2.4 Kutta 格式

进一步用三个点 $x_n,x_{n+\frac{1}{2}},x_{n+1}$ 的斜率值加权平均生成区间 $[x_n,x_{n+1}]$ 上的平均斜率，而考察如下形式的差分格式

$$y_{n+1}=y_n+h(\lambda_0y_n'+\lambda_1y_{n+\frac{1}{2}}'+\lambda_2y_{n+1}')$$

其对应的近似关系式为

$$y(1) \approx y(0) + \lambda_0 y'(0) + \lambda_1 y'\left(\frac{1}{2}\right) + \lambda_2 y'(1)$$

它对 $y=1$ 自然准确，令对于 $y=x, x^2, x^3$ 准确成立，可列出方程组

$$\begin{cases} \lambda_0 + \lambda_1 + \lambda_2 = 1 \\ \lambda_1 + 2\lambda_2 = 1 \\ \dfrac{3}{4}\lambda_1 + 3\lambda_2 = 1 \end{cases}$$

解得

$$\lambda_0 = \lambda_2 = \frac{1}{6}, \lambda_1 = \frac{2}{3}$$

于是有

$$y_{n+1} = y_n + \frac{h}{6}(y_n' + 4y_{n+\frac{1}{2}}' + y_{n+1}')$$

为使这个式子成为三阶差分格式，剩下的问题是如何利用 y_n, y_n' 预报 $y_{n+\frac{1}{2}}'$ 和 y_{n+1}'？再用 Euler 格式有

$$y_{n+\frac{1}{2}} = y_n + \frac{h}{2} y_n'$$

从而有

$$y_{n+\frac{1}{2}}' = f(x_{n+\frac{1}{2}}, y_{n+\frac{1}{2}})$$

进一步预报 y_{n+1}'．考虑到 y_n' 与 $y_{n+\frac{1}{2}}'$ 为已知，线性插值得

$$\begin{aligned} y_{n+1}' &= \frac{x_{n+1} - x_{n+\frac{1}{2}}}{x_n - x_{n+\frac{1}{2}}} y_n' + \frac{x_{n+1} - x_n}{x_{n+\frac{1}{2}} - x_n} y_{n+\frac{1}{2}}' \\ &= -y_n' + 2y_{n+\frac{1}{2}}' \end{aligned}$$

综上所述，这样设计出的差分格式具有形式

$$\begin{cases} y_{n+1} = y_n + \dfrac{h}{6}(K_1 + 4K_2 + K_3) \\ K_1 = f(x_n, y_n) \\ K_2 = f\left(x_{n+\frac{1}{2}}, y_n + \dfrac{h}{2}K_1\right) \\ K_3 = f(x_{n+1}, y_n + h(-K_1 + 2K_2)) \end{cases}$$

这种三阶格式称作 **Kutta 格式**．

3.2.5 四阶经典 Runge－Kutta 格式

继续这一过程，设法在区间 $[x_n, x_{n+1}]$ 内多预报几个点的斜率值，然后将它们

加权平均作为平均斜率，即可以设计出更高精度的单步格式．这类格式统称**Runge－Kutta格式**．实际计算中常用的Runge－Kutta格式是所谓**四阶经典格式**

$$\begin{cases} y_{n+1}=y_n+\frac{h}{6}(K_1+2K_2+2K_3+K_4) \\ K_1=f(x_n,y_n) \\ K_2=f\left(x_{n+\frac{1}{2}},y_n+\frac{h}{2}K_1\right) \\ K_3=f\left(x_{n+\frac{1}{2}},y_n+\frac{h}{2}K_2\right) \\ K_4=f(x_{n+1},y_n+hK_3) \end{cases} \tag{19}$$

这一格式用4个点 $x_n, x_{n+\frac{1}{2}}, x_{n+\frac{1}{2}}, x_{n+1}$（注意点 $x_{n+\frac{1}{2}}$ 复用了一次）的斜率值 K_1,K_2,K_3,K_4 加权平均生成平均斜率，其中 $K_1=f(x_n,y_n)$ 直接求出，然后再依次预报出 K_2,K_3 和 K_4．可以看到，这一格式每一步需4次计算函数值 f．

图3.3描述了经典Runge－Kutta方法的计算流程．

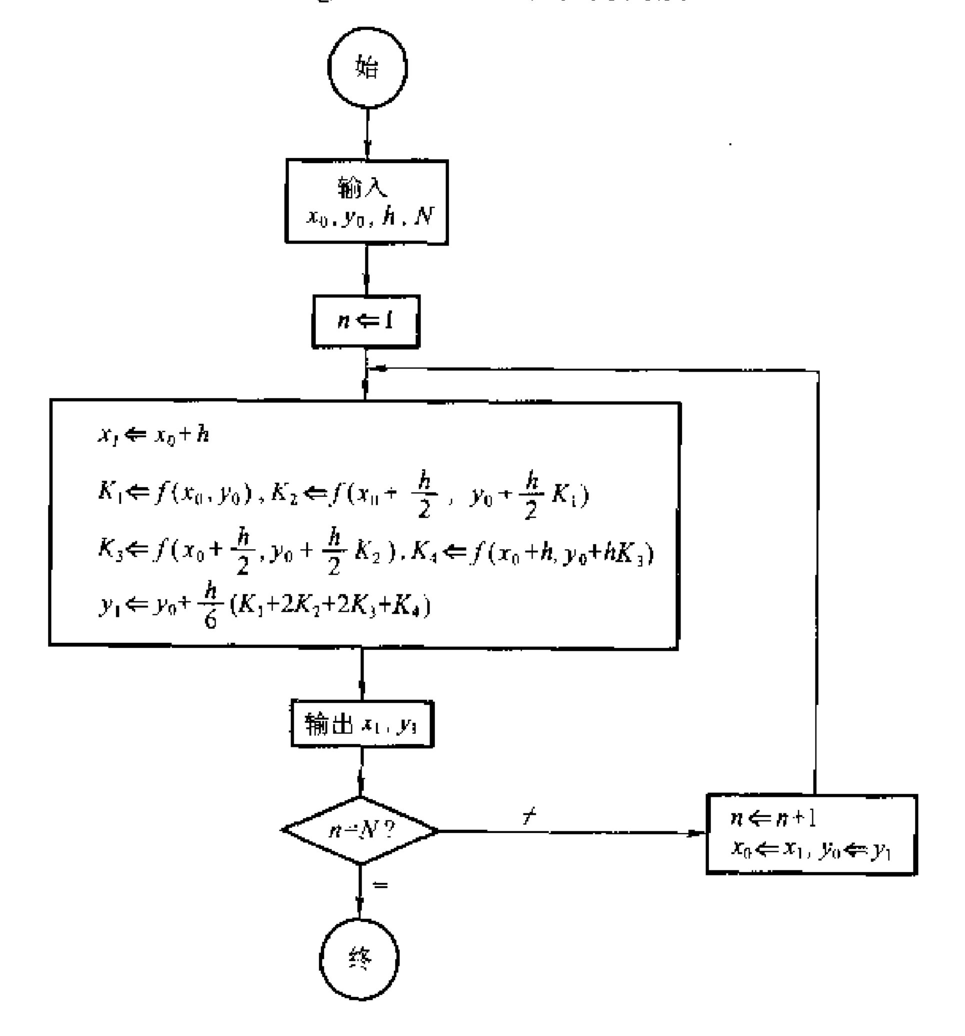


图3.3　四阶经典Runge－Kutta方法的计算流程

例 4 取步长 $h=0.2$，从 $x=0$ 直到 $x=1$ 用四阶经典 Runge－Kutta 格式(19)求解初值问题(4)．

解 这里四阶经典格式(19)中 K_1, K_2, K_3, K_4 的具体形式是

$$K_1 = y_n - \frac{2x_n}{y_n}$$

$$K_2 = y_n + \frac{h}{2}K_1 - \frac{2x_n + h}{y_n + \frac{h}{2}K_1}$$

$$K_3 = y_n + \frac{h}{2}K_2 - \frac{2x_n + h}{y_n + \frac{h}{2}K_2}$$

$$K_4 = y_n + hK_3 - \frac{2(x_n + h)}{y_n + hK_3}$$

表 3.4 记录了计算结果，其中 $y(x_n)$ 仍表示准确解．

表 3.4

x_n	y_n	$y(x_n)$	x_n	y_n	$y(x_n)$
0.2	1.1832	1.1832	0.8	1.6125	1.6125
0.4	1.3417	1.3416	1.0	1.7321	1.7321
0.6	1.4833	1.4832			

比较例 4 与例 2 的计算结果，显然经典格式的精度更高．要注意，虽然经典格式的计算量较改进的 Euler 格式大一倍，但由于这里放大了步长，造出表 3.4 所耗费的计算量几乎与表 3.2 相同．这个例子又一次显示了选择算法的重要意义．

3.3 Adams 方法

前述 Runge－Kutta 方法是一类重要方法，但这类方法的每一步需要先预报几个点上的斜率值，计算量比较大．考虑到在计算 y_{n+1} 之前已得出一系列节点 $x_n, x_{n-1}, \cdots$ 上的斜率值，自然会问，能否利用这些"老信息"来减少计算量呢？这就是 Adams 方法的设计思想．

特别地，Euler 格式

$$\begin{cases} y_{n+1} = y_n + hy_n' \\ y_n' = f(x_n, y_n) \end{cases}$$

和隐式 Euler 格式

$$\begin{cases} y_{n+1} = y_n + h y'_{n+1} \\ y'_{n+1} = f(x_{n+1}, y_{n+1}) \end{cases}$$

是一阶 **Adams 方法**.

3.3.1 二阶 Adams 格式

设用 x_n, x_{n-1} 两点的斜率值加权平均生成区间$[x_n, x_{n+1}]$上的平均斜率,而设计如下形式的差分格式

$$\begin{cases} y_{n+1} = y_n + h[(1-\lambda)y'_n + \lambda y'_{n-1}] \\ y'_n = f(x_n, y_n) \\ y'_{n-1} = f(x_{n-1}, y_{n-1}) \end{cases}$$

现在适当选取参数 λ,使上述格式具有二阶精度.为此考察其对应的近似关系式,仍设 $x_n = 0, h = 1$,这里有

$$y(1) \approx y(0) + (1-\lambda)y'(0) + \lambda y'(-1)$$

令对于 $y = x^2$ 准确可定出 $\lambda = -\frac{1}{2}$,这样设计出的计算格式

$$y_{n+1} = y_n + \frac{h}{2}(3y'_n - y'_{n-1})$$

称作**二阶显式 Adams 格式**.

类似地,改用 x_n, x_{n+1} 两个节点的斜率值 y'_n 与 y'_{n+1} 生成区间$[x_n, x_{n+1}]$上的平均斜率,而使格式

$$\begin{cases} y_{n+1} = y_n + h[(1-\lambda)y'_{n+1} + \lambda y'_n] \\ y'_n = f(x_n, y_n) \\ y'_{n+1} = f(x_{n+1}, y_{n+1}) \end{cases}$$

具有二阶精度,不难定出 $\lambda = \frac{1}{2}$,从而有**二阶隐式 Adams 格式**

$$y_{n+1} = y_n + \frac{h}{2}(y'_{n+1} + y'_n)$$

它是熟知的梯形格式(8).

3.3.2 误差的事后估计

仿照改进的 Euler 格式的构造方法,可以将显式与隐式两种 Adams 格式匹配在一起,构成下列**二阶 Adams 预报校正系统**:

$$\text{预报} \quad \begin{aligned} \bar{y}_{n+1} &= y_n + \frac{h}{2}(3y'_n - y'_{n-1}) \\ \bar{y}'_{n+1} &= f(x_{n+1}, \bar{y}_{n+1}) \end{aligned} \qquad (20)$$

校正 $$y_{n+1}=y_n+\frac{h}{2}(\bar{y}'_{n+1}+y'_n)$$

$$y'_{n+1}=f(x_{n+1},y_{n+1})$$

这种预报校正系统是个两步法，它在计算 y_{n+1} 时不但要用到前一步的信息 y_n, y'_n，而且要用到更前一步的信息 y'_{n-1}，因此它不能自行启动. 在实际计算时，可以先借助于某种单步法——譬如具有二阶精度的改进的 Euler 格式(13)提供开始值 y_1，然后再启动上述预报校正系统逐步计算下去.

上述预报校正技术不仅能设计出实用算法，而且还能用于误差的事后估计. 为此再考察系统(20)中预报与校正两种格式

$$p_{n+1}=y_n+\frac{h}{2}(3y'_n-y'_{n-1})$$

$$c_{n+1}=y_n+\frac{h}{2}(y'_{n+1}+y'_n)$$

注意到它们均具有二阶精度，进一步将它们加工成具有三阶精度的计算格式

$$y_{n+1}=(1-\omega)p_{n+1}+\omega c_{n+1}$$

为此考察其对应的近似关系式

$$y(x_{n+1})\approx y(x_n)+(1-\omega)\frac{h}{2}[3y'(x_n)-y'(x_{n-1})]+$$

$$\omega\frac{h}{2}[y'(x_{n+1})+y'(x_n)]$$

不妨设 $x_n=0, h=1$，令对于 $y=x^3$ 准确成立，可定出 $\omega=\frac{5}{6}$，从而有

$$y_{n+1}=\frac{1}{6}p_{n+1}+\frac{5}{6}c_{n+1}$$

由于这里 y_{n+1} 是具有三阶精度的"准确"值，因而可以用预报值与校正值两者的偏差来估计它们的误差：

$$y_{n+1}-p_{n+1}=-\frac{5}{6}(p_{n+1}-c_{n+1})$$
$$y_{n+1}-c_{n+1}=\frac{1}{6}(p_{n+1}-c_{n+1}) \tag{21}$$

利用误差作为计算结果的一种补偿有可能改善精度，因而基于这种误差的事后估计可以进一步优化预报校正系统(20). 就是说，按式(21)，$p_{n+1}-\frac{5}{6}(p_{n+1}-c_{n+1})$ 与 $c_{n+1}+\frac{1}{6}(p_{n+1}-c_{n+1})$ 分别可以看作 p_{n+1} 与 c_{n+1} 的改进值. 在校正值 c_{n+1} 求出之前，自然用上一步的偏差值 p_n-c_n 替代 $p_{n+1}-c_{n+1}$ 进行计算，这样，系统(20)可修改为如下**改进的二阶 Adams 预报校正系统**：

$$
\begin{aligned}
&\text{预报} \quad p_{n+1} = y_n + \frac{h}{2}(3y_n' - y_{n-1}') \\
&\text{改进} \quad m_{n+1} = p_{n+1} - \frac{5}{6}(p_n - c_n) \\
&\qquad\quad\ m_{n+1}' = f(x_{n+1}, m_{n+1}) \\
&\text{校正} \quad c_{n+1} = y_n + \frac{h}{2}(m_{n+1}' + y_n') \\
&\text{改进} \quad y_{n+1} = c_{n+1} + \frac{1}{6}(p_{n+1} - c_{n+1}) \\
&\qquad\quad\ y_{n+1}' = f(x_{n+1}, y_{n+1})
\end{aligned} \tag{22}
$$

需要指出的是,运用上述计算方案时要用到前面两步的信息 $y_n, y_n', y_{n-1}, y_{n-1}'$ 和 $p_n - c_n$,因此在启动之前必须先提供开始值 y_1 与 $p_1 - c_1$. 同 Adams 预报校正系统(20)一样,开始值 y_1 可用改进的 Euler 格式(9)来提供,而 $p_1 - c_1$ 一般令其等于 0.

3.3.3　实用的四阶 Adams 预报校正系统

运用上述处理方法,不难导出如下显式与隐式**四阶 Adams 格式**

$$
\begin{aligned}
y_{n+1} &= y_n + \frac{h}{24}(55y_n' - 59y_{n-1}' + 37y_{n-2}' - 9y_{n-3}') \\
y_{n+1} &= y_n + \frac{h}{24}(9y_{n+1}' + 19y_n' - 5y_{n-1}' + y_{n-2}')
\end{aligned} \tag{23}
$$

将两者匹配在一起,即可生成下列**四阶 Adams 预报校正系统**

$$
\begin{aligned}
&\text{预报} \quad \bar{y}_{n+1} = y_n + \frac{h}{24}(55y_n' - 59y_{n-1}' + 37y_{n-2}' - 9y_{n-3}') \\
&\qquad\quad\ \bar{y}_{n+1}' = f(x_{n+1}, \bar{y}_{n+1}) \\
&\text{校正} \quad y_{n+1} = y_n + \frac{h}{24}(9\bar{y}_{n+1}' + 19y_n' - 5y_{n-1}' + y_{n-2}') \\
&\qquad\quad\ y_{n+1}' = f(x_{n+1}, y_{n+1})
\end{aligned} \tag{24}
$$

这种**四阶 Adams 预报校正系统**是个四步法,它在计算 y_{n+1} 时不但要用到前一步的信息 y_n, y_n',而且要用到更前面三步的信息 $y_{n-1}', y_{n-2}', y_{n-3}'$,因此它不能自行启动. 实际计算时,需要借助于某种单步法——譬如四阶经典的 Runge－Kutta 格式(19)为其提供开始值 y_1, y_2, y_3.

图 3.4 描述了这种算法的计算流程.

例 5　用四阶 Adams 预报校正系统(24)求解初值问题(4).

解　取步长 $h = 0.1$,用四阶 Runge－Kutta 格式(19)提供开始值,然后套用四阶 Adams 系统(24)逐步计算. 计算结果见表 3.5. 表中 $\bar{y}_n$ 和 y_n 分别为预报值

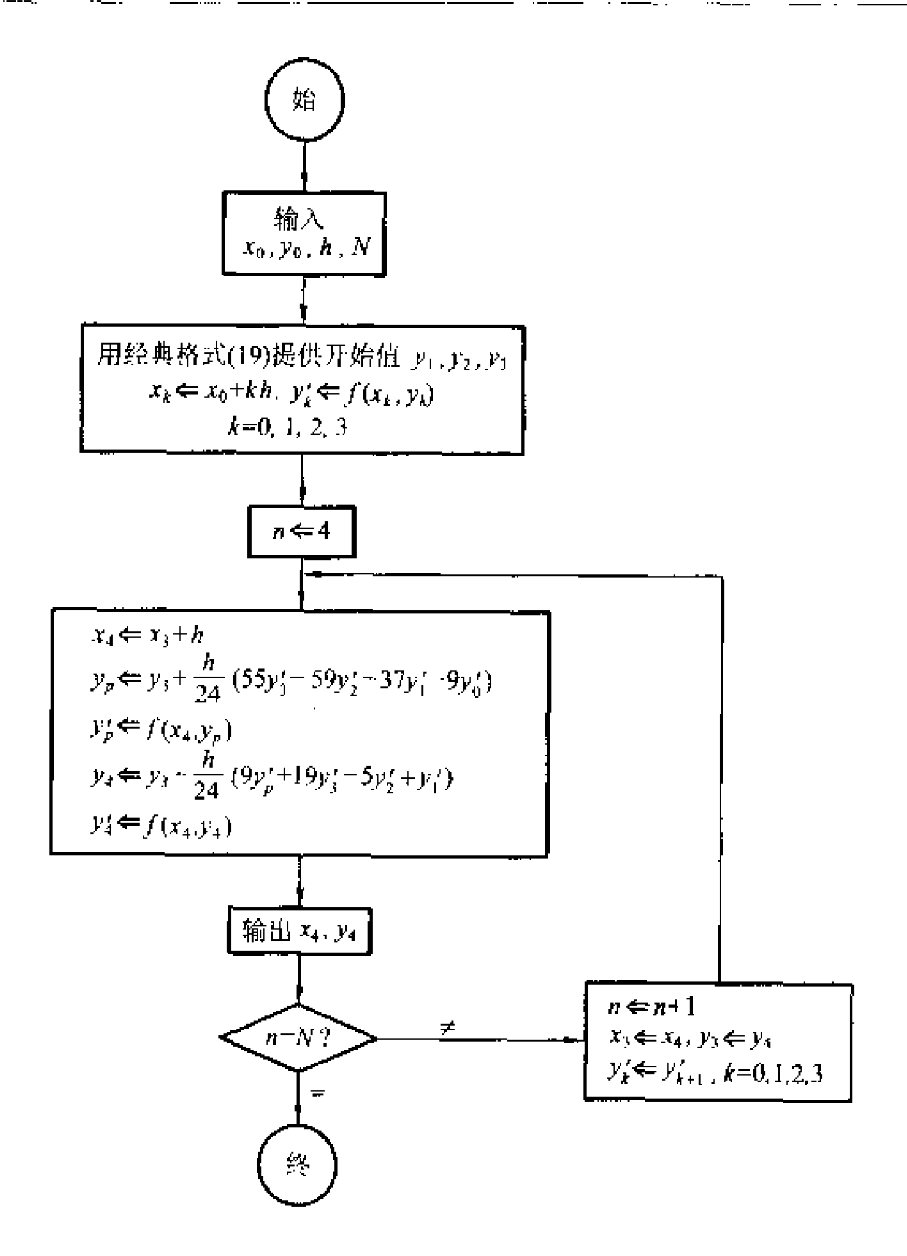


图 3.4 四阶 Adams 预报校正系统的计算流程

与校正值，同时列出了准确值 $y(x_n)$ 以显示计算结果的精度.

表 3.5

x_n	$\bar{y}_n$	y_n	$y(x_n)$
0.0		1.000 0	1.000 0
0.1		1.095 4	1.095 4
0.2		1.183 2	1.183 2
0.3		1.264 9	1.264 9
0.4	1.341 5	1.341 6	1.341 6
0.5	1.414 1	1.414 2	1.414 2
0.6	1.483 2	1.483 2	1.483 2

续表

x_n	y_n	y_n	$y(x_n)$
0.7	1.549 1	1.549 2	1.549 2
0.8	1.612 4	1.612 4	1.612 5
0.9	1.673 3	1.673 3	1.673 3
1.0	1.732 0	1.732 0	1.732 1

仿照四阶 Adams 格式的处理方法估计系统(24)中预报值 p_{n+1} 与校正值 c_{n+1} 的误差. 为此，考察如下形式的 5 阶格式

$$y_{n+1}=(1-\omega)p_{n+1}+\omega c_{n+1}$$

不难定出

$$\omega=\frac{251}{270}$$

从而有误差估计式

$$
\begin{aligned}
y_{n+1}-p_{n+1}&=-\frac{251}{270}(p_{n+1}-c_{n+1})\\
y_{n+1}-c_{n+1}&=\frac{19}{270}(p_{n+1}-c_{n+1})
\end{aligned}
\tag{25}
$$

利用这一误差估计式修改四阶 Adams 预报校正系统(24)，即可导出下列改进的**四阶 Adams 预报校正系统：**

$$
\begin{aligned}
&\text{预报}\quad p_{n+1}=y_n+\frac{h}{24}(55y_n'-59y_{n-1}'+37y_{n-2}'-9y_{n-3}')\\
&\text{改进}\quad m_{n+1}=p_{n+1}+\frac{251}{270}(c_n-p_n)\\
&\qquad\quad m_{n+1}'=f(x_{n+1},m_{n+1})\\
&\text{校正}\quad c_{n+1}=y_n+\frac{h}{24}(9m_{n+1}'+19y_n'-5y_{n-1}'+y_{n-2}')\\
&\text{改进}\quad y_{n+1}=c_{n+1}+\frac{19}{270}(p_{n+1}-c_{n+1})\\
&\qquad\quad y_{n+1}'=f(x_{n+1},y_{n+1})
\end{aligned}
$$

3.4 收敛性与稳定性

3.4.1 收敛性问题

前面已看到，差分方法的设计思想是，通过离散化手续将微分方程化归为差分方程（代数方程）来求解. 这种转化是否合适，要看差分方程的解 y_n 当 $h\to 0$ 时

是否收敛到微分方程的准确解 $y(x_n)$.

定义 2 对于任给 $x_n = x_0 + nh$，如果数值解 y_n 当 $h \to 0$（同时 $n \to \infty$）时趋向于准确解 $y(x_n)$，则称该差分方法是**收敛**的.

收敛性问题比较复杂.为解释收敛性的含义，这里仅考察下列模型问题

$$\begin{cases} y' = \lambda y, & \lambda < 0 \\ y(0) = y_0 \end{cases} \tag{26}$$

这个问题有准确解

$$y = y_0 e^{\lambda t}$$

先考察 Euler 格式的收敛性.问题(26)的 Euler 格式具有形式

$$y_{n+1} = (1 + h\lambda) y_n \tag{27}$$

从而数值解

$$\begin{aligned} y_n &= (1 + h\lambda)^n y_0 \\ &= y_0 \left[(1 + h\lambda)^{\frac{1}{h\lambda}} \right]^{nh\lambda} \\ &= y_0 \left[(1 + h\lambda)^{\frac{1}{h\lambda}} \right]^{\lambda x_n} \end{aligned}$$

因之当 $h \to 0$ 时

$$y_n \to y_0 e^{\lambda x_n} = y(x_n)$$

可见问题(26)的 Euler 格式是收敛的.

再考察隐式 Euler 格式.问题(26)的隐式 Euler 格式为

$$y_{n+1} = y_n + h\lambda y_{n+1} \tag{28}$$

这时有

$$y_{n+1} = \frac{1}{1 - h\lambda} y_n$$

从而数值解

$$\begin{aligned} y_n &= y_0 \left(\frac{1}{1 - h\lambda} \right)^n \\ &= y_0 \left[\left(1 + \frac{h\lambda}{1 - h\lambda} \right)^{\frac{1}{h\lambda} \cdot \frac{h\lambda}{\ }} \right]^{\frac{nh\lambda}{1 - h\lambda}} \end{aligned}$$

这时当 $h \to 0$ 时仍然有 $y_n \to y(x_n)$，因而问题(26)的隐式 Euler 格式同样是收敛的.

3.4.2 稳定性问题

前面关于收敛性的讨论有个前提，必须假定差分方法的每一步计算都是准确的.实际情形并不是这样，差分方程的求解还会有计算误差，譬如由于数字舍入而引起的扰动.这类扰动在传播过程中会不会恶性增长，以至于"淹没"了差分

方程的"真解"呢？这就是差分方法的稳定性问题.

在实际计算时，人们希望某一步所产生的扰动值在后面的计算中能够被抑制，甚至是逐步衰减的. 具体地说：

定义 3 如果一种差分方法在节点值 y_n 上大小为 δ 的扰动，导致以后各节点值 $y_m(m>n)$ 上产生的偏差均不超过 δ，则称该方法是**稳定**的.

再针对问题(26)考察 Euler 格式的稳定性. 设在节点值 y_n 上有一扰动值 ε_n，它的传播使节点值 y_{n+1} 上产生大小为 ε_{n+1} 的扰动值，假设 Euler 格式(27)的计算过程不再引进新的误差，则扰动值满足

$$\varepsilon_{n+1}=(1+h\lambda)\varepsilon_n$$

可见扰动值满足原来的差分方程(27). 这样，如果原差分方程的解是不增长的，即有

$$|y_{n+1}|\leqslant|y_n|$$

这时就能保证格式的稳定性.

显然，为要保证差分方程(27)的解不增长，必须选取 h 充分小，使

$$|1+h\lambda|\leqslant 1$$

这表明 Euler 格式是**条件稳定**的. 上述稳定性条件亦可表为

$$h\leqslant-\frac{2}{\lambda}$$

再考察隐式 Euler 格式(28)，由于 $\lambda<0$，这时恒成立

$$\left|\frac{1}{1-h\lambda}\right|\leqslant 1$$

从而总有 $|y_{n+1}|\leqslant|y_n|$，这说明隐式 Euler 格式是**恒稳定(无条件稳定)**的.

3.5 方程组与高阶方程的情形

3.5.1 一阶方程组

前面研究了单个方程 $y'=f$ 的差分方法，只要把 y 和 f 理解为向量，所提供的各种算法即可推广应用到一阶方程组的情形.

譬如，对于方程组

$$\begin{cases}y'=f(x,y,z), & y(x_0)=y_0\\ z'=g(x,y,z), & z(x_0)=z_0\end{cases}$$

引进节点 $x_n=x_0+nh$，$n=1,2,\cdots$，以 y_n，z_n 表示节点 x_n 上的近似解，则其改进的 Euler 格式具有形式：

预报　$y_{n+1}=y_n+hf(x_n,y_n,z_n)$

$$z_{n+1}=z_n+hg(x_n,y_n,z_n)$$

校正　$y_{n+1}=y_n+\frac{h}{2}[f(x_n,y_n,z_n)+f(x_{n+1},\bar{y}_{n+1},z_{n+1})]$

$$z_{n+1}=z_n+\frac{h}{2}[g(x_n,y_n,z_n)+g(x_{n+1},y_{n+1},\bar{z}_{n+1})]$$

而其四阶 Runge－Kutta 格式(经典格式)则为

$$\begin{cases} y_{n+1}=y_n+\frac{h}{6}(K_1+2K_2+2K_3+K_4) \\ z_{n+1}=z_n+\frac{h}{6}(L_1+2L_2+2L_3+L_4) \end{cases} \tag{29}$$

式中

$$\begin{aligned} K_1&=f(x_n,y_n,z_n)\\ L_1&=g(x_n,y_n,z_n)\\ K_2&=f\left(x_{n+\frac{1}{2}},y_n+\frac{h}{2}K_1,z_n+\frac{h}{2}L_1\right)\\ L_2&=g\left(x_{n+\frac{1}{2}},y_n+\frac{h}{2}K_1,z_n+\frac{h}{2}L_1\right)\\ K_3&=f\left(x_{n+\frac{1}{2}},y_n+\frac{h}{2}K_2,z_n+\frac{h}{2}L_2\right)\\ L_3&=g\left(x_{n+\frac{1}{2}},y_n+\frac{h}{2}K_2,z_n+\frac{h}{2}L_2\right)\\ K_4&=f(x_{n+1},y_n+hK_3,z_n+hL_3)\\ L_4&=g(x_{n+1},y_n+hK_3,z_n+hL_3) \end{aligned} \tag{30}$$

这里四阶 Runge－Kutta 格式依然是一步法，利用节点值 y_n,z_n 按式(30)顺序计算 $K_1,L_1,K_2,L_2,K_3,L_3,K_4,L_4$，然后代入式(29)即可求得节点值 y_{n+1}，z_{n+1}.

3.5.2　化高阶方程为一阶方程组

关于高阶微分方程(或方程组)的初值问题，原则上总可以归结为一阶方程组来求解.譬如，对于下列二阶方程的初值问题：

$$\begin{cases} y''=f(x,y,y') \\ y(x_0)=y_0,\ y'(x_0)=y_0' \end{cases}$$

若引进新的变量 $z=y'$ 即可化归为一阶方程组的初值问题：

$$\begin{cases} y'=z, & y(x_0)=y_0 \\ z'=f(x,y,z), & z(x_0)=y_0' \end{cases}$$

针对这个问题应用四阶 Runge－Kutta 格式(29)有

$$\begin{cases} y_{n+1}=y_n+\dfrac{h}{6}(K_1+2K_2+2K_3+K_4) \\ z_{n+1}=z_n+\dfrac{h}{6}(L_1+2L_2+2L_3+L_4) \end{cases}$$

按式(30)，有

$$\begin{aligned} &K_1=z_n, && L_1=f(x_n,y_n,z_n) \\ &K_2=z_n+\frac{h}{2}L_1, && L_2=f\left(x_{n+\frac{1}{2}},y_n+\frac{h}{2}K_1,z_n+\frac{h}{2}L_1\right) \\ &K_3=z_n+\frac{h}{2}L_2, && L_3=f\left(x_{n+\frac{1}{2}},y_n+\frac{h}{2}K_2,z_n+\frac{h}{2}L_2\right) \\ &K_4=z_n+hL_3, && L_4=f(x_{n+1},y_n+hK_3,z_n+hL_3) \end{aligned}$$

消去 K_1,K_2,K_3,K_4，上述格式化简为

$$\begin{cases} y_{n+1}=y_n+hz_n+\dfrac{h^2}{6}(L_1+L_2+L_3) \\ z_{n+1}=z_n+\dfrac{h}{6}(L_1+2L_2+2L_3+L_4) \end{cases}$$

式中

$$\begin{aligned} &L_1=f(x_n,y_n,z_n) \\ &L_2=f\left(x_{n+\frac{1}{2}},y_n+\frac{h}{2}z_n,z_n+\frac{h}{2}L_1\right) \\ &L_3=f\left(x_{n+\frac{1}{2}},y_n+\frac{h}{2}z_n+\frac{h^2}{4}L_1,z_n+\frac{h}{2}L_2\right) \\ &L_4=f\left(x_{n+1},y_n+hz_n+\frac{h^2}{2}L_2,z_n+hL_3\right) \end{aligned}$$

3.6 边值问题

在具体求解微分方程时，必须附加某种定解条件．微分方程和定解条件一起组成**定解问题**．对高阶常微分方程，定解条件通常有两种给法：一种是给出积分曲线在初始时刻的性态，这类条件称**初始条件**，相应的定解问题就是前面已讨论过的初值问题；另一种是给出了积分曲线首末两端的性态，这类定解条件称**边界条件**，相应的定解问题称为**边值问题**．

譬如，考察下列二阶线性方程的边值问题：

$$\begin{cases} y''+p(x)y'+q(x)y=r(x), \quad a<x<b \\ y(a)=\alpha, \quad y(b)=\beta \end{cases} \tag{31}$$

为要应用差分法，关键在于恰当地选取差商逼近微分方程中的导数项．令

$$y'(x)\approx\frac{y(x+h)-y(x-h)}{2h}$$

$$y''(x)\approx\frac{y(x+h)-2y(x)+y(x-h)}{h^2}$$

设将求解区间$[a,b]$划分为 N 等分,步长 $h=\frac{b-a}{N}$,节点 $x_n=x_0+nh$, $n=0,1,\cdots,N$,用差商替代相应的导数,可将边值问题(31)离散化,导出下列差分方程组

$$\begin{cases}\frac{y_{n+1}-2y_n+y_{n-1}}{h^2}+p_n\frac{y_{n+1}-y_{n-1}}{2h}+q_ny_n=r_n, & n=1,2,\cdots,N-1\\ y_0=\alpha, \quad y_N=\beta\end{cases}$$

式中 p_n,q_n,r_n的下标 n 表示在节点 x_n处取值.从上面的式子中消去已知的 y_0 和 y_N,可整理得到关于 y_n的下列方程组

$$\begin{cases}(-2+h^2q_1)y_1+\left(1+\frac{h}{2}p_1\right)y_2=h^2r_1-\left(1-\frac{h}{2}p_1\right)\alpha\\ \left(1-\frac{h}{2}p_n\right)y_{n-1}+(-2+h^2q_n)y_n+\left(1+\frac{h}{2}p_n\right)y_{n+1}=h^2r_n,\\ \qquad\qquad n=2,3,\cdots,N-2\\ \left(1-\frac{h}{2}p_{N-1}\right)y_{N-2}+(-2+h^2q_{N-1})y_{N-1}=h^2r_{N-1}-\left(1+\frac{h}{2}p_{N-1}\right)\beta\end{cases}\tag{32}$$

这样归结出的方程组是三对角型的,求解这类方程组可用追赶法(参看第六章6.1节).

小　结

微分方程是个连续的计算模型,它要求给出函数形式的解 $y(x)$,并且方程中含有极限化的导数项 $y'(x)$,因此用计算机处理这类问题必须将计算模型离散化.

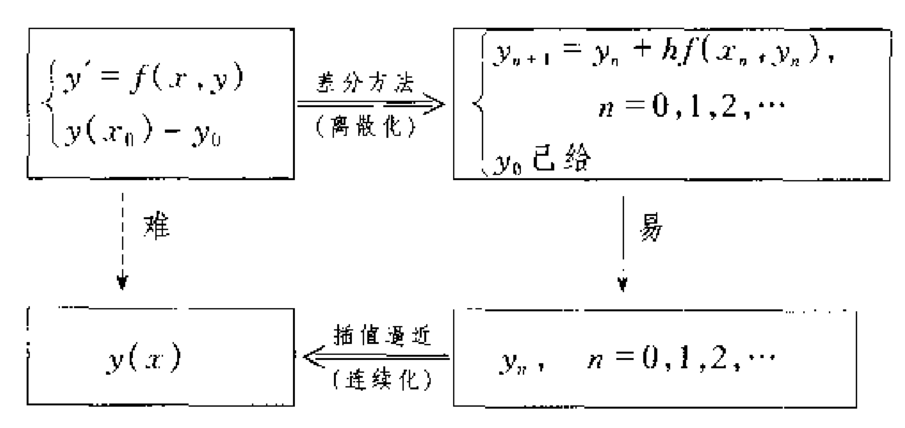
为此,首先将解 $y(x)$表达为数据表(x_n,y_n), $n=0,1,2,\cdots$的形式,从而将分析问题化归为决定参数值 y_n的代数问题.问题在于所归结出的代数问题的规模往往很大.对于常微分方程的初值问题,可以采取"逐步推进"的求解方式顺序计算 $y_0\to y_1\to y_2\to\cdots\to y_n\to\cdots$.这样,每一计算步只要求出一个近似值 y_{n+1},这就大大地缩减了计算模型的规模.

差分格式分显式与隐式两大类,它们各有所长.显式格式计算量小,但稳定性差.与此相反,隐式格式的稳定性强,但计算困难.在差分方法设计时,预报校正技术综合了显式与隐式两种格式的优势.

差分格式的设计追求高精度.用低阶格式生成高阶格式,一个有效的途径是运用松弛技术.

总之,缩减技术、校正技术与松弛技术,算法设计的基本技术对差分方法的设计同样有指导意义.

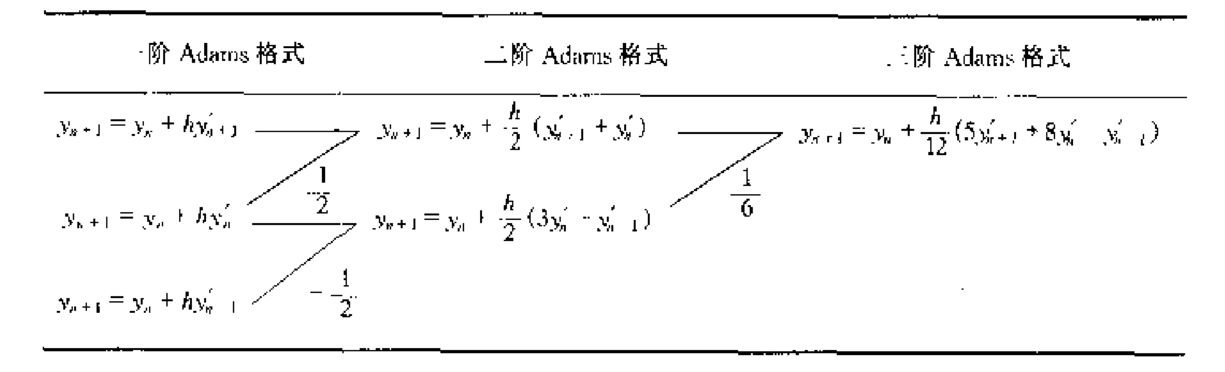
最后再概括差分方法的设计思想.求解常微分方程的初值问题 $y'=f(x,y), y(x_0)=y_0$,是要寻求某个函数 $y(x)$,它满足所给方程即恒成立 $y'(x)\equiv f(x,y(x))$,且在点 x_0 取给定初值 y_0,从数学分析的角度求解这个问题往往是困难的.差分方法将这个分析问题转化为计算离散值 $\{y_n\}$ 的代数问题,而依据所获得的一系列离散值 y_n,通过插值逼近方法,容易构造出 $y(x)$ 的某个近似表达式(参看第一章).可以用图刻画这一处理过程.



题解 3.1　Adams 格式的设计

提要　Adams 格式的设计基于代数精度的概念,其具体设计方法有两条可供选择的途径:直接化归为求解线性方程组,或者回避求解方程组,运用松弛技术从低阶到高阶逐步生成.

运用松弛技术逐步生成各阶 Adams 格式,其加工过程如下所示:



其中符号的具体含义是

$$a \text{---} c = (1-\omega)a + \omega b$$
$$\omega$$
$$b$$

下列各题将逐步证明这一事实.

题 1 利用 y_n', y_{n-1}' 设计差分格式

$$y_{n+1} = y_n + h(\lambda_0 y_n' + \lambda_1 y_{n-1}')$$

解 令 $n=0, h=1$,考察其对应的近似关系式

$$y(1) \approx y(0) + \lambda_0 y'(0) + \lambda_1 y'(-1)$$

它对于 $y=1$ 自然准确,令对于 $y=x, y=x^2$ 准确成立,可列出方程组

$$\begin{cases} \lambda_0 + \lambda_1 = 1 \\ -2\lambda_1 = 1 \end{cases}$$

据此定出

$$\lambda_0 = \frac{3}{2}, \quad \lambda_1 = -\frac{1}{2}$$

从而所要设计的差分格式为

$$y_{n+1} = y_n + \frac{h}{2}(3y_n' - y_{n-1}')$$

它是二阶显式 Adams 格式.

题 2 利用两种 Euler 格式

$$y_{n+1,1} = y_n + hy_n'$$
$$y_{n+1,2} = y_n + hy_{n-1}'$$

松弛生成高精度的差分格式

$$y_{n+1} = (1-\omega)y_{n+1,1} + \omega y_{n+1,2}$$

解 由于 Euler 格式 $y_{n+1,1}$ 与 $y_{n+1,2}$ 均有一阶精度,不管松弛因子 ω 如何选择,上述差分格式至少有一阶精度.考察所求格式

$$y_{n+1} = (1-\omega)(y_n + hy_n') + \omega(y_n + hy_{n-1}')$$

所对应的近似关系式

$$y(1) \approx (1-\omega)y'(0) + \omega y'(-1)$$

令对于 $y=x^2$ 准确成立,则可列出方程

$$-2\omega = 1$$

从而有 $\omega = -\dfrac{1}{2}$,这样设计出的差分格式

$$y_{n+1} = \frac{3}{2}y_{n+1,1} - \frac{1}{2}y_{n+1,2}$$

具有形式

$$y_{n+1}=y_n+\frac{h}{2}(3y_n'-y_{n-1}')$$

它正是题 1 设计出的二阶显式 Adams 格式.

题 3　利用 3 个斜率信息 y_{n+1}', y_n', y_{n-1}'，设计差分格式

$$y_{n+1}=y_n+h(\lambda_{-1}y_{n+1}'+\lambda_0 y_n'+\lambda_1 y_{n-1}')$$

解　考察其对应的近似关系式

$$y(1)\approx y(0)+\lambda_{-1}y'(1)+\lambda_0 y'(0)+\lambda_1 y'(-1)$$

它对于 $y=1$ 自然准确，令对于 $y=x,x^2,x^3$ 准确成立，可列出方程组

$$\begin{cases}\lambda_{-1}+\lambda_0+\lambda_1=1\\ \lambda_{-1}-\lambda_1=\dfrac{1}{2}\\ \lambda_{-1}+\lambda_1=\dfrac{1}{3}\end{cases}$$

据此定出

$$\lambda_{-1}=\frac{5}{12},\quad \lambda_0=\frac{8}{12}=\frac{2}{3},\quad \lambda_1=-\frac{1}{12}$$

从而设计出三阶隐式 Adams 格式

$$y_{n+1}=y_n+\frac{h}{12}(5y_{n+1}'+8y_n'-y_{n-1}')$$

题 4　利用二阶隐式 Adams 格式(梯形格式)

$$y_{n+1,1}=y_n+\frac{h}{2}(y_{n+1}'+y_n')$$

与二阶显式 Adams 格式(见题 1)

$$y_{n+1,2}=y_n+\frac{h}{2}(3y_n'-y_{n-1}')$$

松弛生成更高精度的差分格式

$$y_{n+1}=(1-\omega)y_{n+1,1}+\omega y_{n+1,2}$$

解　所求格式对应的近似关系式为

$$y(1)\approx\frac{1-\omega}{2}[y'(1)+y'(0)]+\frac{\omega}{2}[3y'(0)-y'(-1)]$$

令对于 $y=x^3$ 准确成立即定出 $\omega=\frac{1}{6}$，从而有

$$\begin{aligned}y_{n+1}&=\frac{5}{6}y_{n+1,1}+\frac{1}{6}y_{n+1,2}\\&=y_n+\frac{h}{12}(5y_{n+1}'+8y_n'-y_{n-1}')\end{aligned}$$

据此同样设计出三阶隐式 Adams 格式(试与题 3 相比较).

题解 3.2　线性多步法

提要　在差分方法逐步推进的求解过程中，计算 y_{n+1} 之前事实上已经求出了一系列近似值 $y_n, y_n', y_{n-1}, y_{n-1}', \cdots$. 如果充分利用前面多步的信息来计算 y_{n+1}，则可以期望以较小的代价获得较高的精度，这就是线性多步法的设计思想. 前已介绍过的 Adams 方法是一类特殊的线性多步法.

线性多步格式的设计方法依然是，基于代数精度的概念，将问题归结为求解某个线性方程组.

题 1　设计下列形式的差分格式：

$$y_{n+1} = ay_n + by_{n-1} + h(cy_n' + dy_{n-1}')$$

解　令 $n=0, h=1$，考察对应的近似关系式

$$y(1) \approx ay(0) + by(-1) + cy'(0) + dy'(-1)$$

注意到式中含有 4 个待定参数，令它对于 $y=1, x, x^2, x^3$ 准确成立，可列出方程组

$$\begin{cases} a+b=1 \\ -b+c+d=1 \\ b-2d=1 \\ -b+3d=1 \end{cases}$$

解之得

$$a=-4,\ b=5,\ c=4,\ d=2$$

这样设计出的差分格式是

$$y_{n+1} = -4y_n + 5y_{n-1} + 2h(2y_n' + y_{n-1}')$$

它有三阶精度.

题 2　确定参数 a 的值，使下列格式有 4 阶精度：

$$y_{n+1} = a(y_n - y_{n-1}) + y_{n-2} + \frac{h}{2}(3-a)(y_n' + y_{n-1}')$$

解　令 $n=0, h=1$，考察其对应的近似关系式

$$y(1) \approx a[y(0) - y(-1)] + y(-2) + \frac{1}{2}(3-a)[y'(0) + y'(-1)]$$

为保证所要设计的格式有 4 阶精度，令它对于 $y=1, x, x^2, x^3, x^4$ 准确成立，据此列出的方程可唯一地定出 $a=-9$，从而所求的 4 阶格式为

$$y_{n+1} = -9(y_n - y_{n-1}) + y_{n-2} + 6h(y_n' + y_{n-1}')$$

题 3　设计下列形式的差分格式：

$$y_{n+1} = y_{n-1} + 2h(\lambda_0 y_{n+1}' + \lambda_1 y_n' + \lambda_2 y_{n-1}')$$

解　令 $n=0, h=1$,考察其对应的近似关系式

$$y(1)\approx y(-1)+2[\lambda_0 y'(1)+\lambda_1 y'(0)+\lambda_2 y'(-1)]$$

它对于 $y=1$ 自然准确.令它对于 $y=x, x^2, x^3$ 准确成立,可列出方程组

$$\begin{cases}\lambda_0+\lambda_1+\lambda_2=1\\ \lambda_0-\lambda_2=0\\ \lambda_0+\lambda_2=\dfrac{1}{3}\end{cases}$$

据此定出

$$\lambda_0=\lambda_2=\frac{1}{6},\quad \lambda_1=\frac{2}{3}$$

这样设计出的差分格式

$$y_{n+1}=y_{n-1}+\frac{h}{3}(y'_{n+1}+4y'_n+y'_{n-1})$$

称作 **Simpson 格式**,它是个隐式格式,实际上有 4 阶精度.

题 4　设计下列形式的差分格式:

$$y_{n+1}=y_{n-3}+4h(\lambda_0 y'_n+\lambda_1 y'_{n-1}+\lambda_2 y'_{n-2})$$

解　令 $n=0, h=1$,考察其对应的近似关系式

$$y(1)\approx y(-3)+4[\lambda_0 y'(0)+\lambda_1 y'(-1)+\lambda_2 y'(-2)]$$

它对于 $y=1$ 显然准确.令对于 $y=x, x^2, x^3$ 准确成立,可列出方程组

$$\begin{cases}\lambda_0+\lambda_1+\lambda_2=1\\ \lambda_1+2\lambda_2=1\\ \lambda_1+4\lambda_2=\dfrac{7}{3}\end{cases}$$

据此定出

$$\lambda_0=\frac{2}{3},\quad \lambda_1=-\frac{1}{3},\quad \lambda_2=\frac{2}{3}$$

这样设计出的差分格式

$$y_{n+1}=y_{n-3}+\frac{4h}{3}(2y'_n-y'_{n-1}+2y'_{n-2})$$

称作 **Milne 格式**.它是个显式格式,实际上有 4 阶精度.

题 5　设计下列形式的差分格式:

$$y_{n+1}=\mu_0 y_n+\mu_1 y_{n-2}+h(\lambda_0 y'_{n+1}+\lambda_1 y'_n+\lambda_2 y'_{n-1})$$

解　令它对于 $y=1, x, x^2, x^3, x^4$ 准确成立,可列出方程组

$$\begin{cases}\mu_0+\mu_1=1\\ -2\mu_1+\lambda_0+\lambda_1+\lambda_2=1\\ 4\mu_1+2(\lambda_0-\lambda_2)=1\\ -8\mu_1+3(\lambda_0+\lambda_2)=1\\ 16\mu_1+4(\lambda_0-\lambda_2)=1\end{cases}$$

据此定出

$$\mu_0=\frac{9}{8},\quad \mu_1=-\frac{1}{8}$$

$$\lambda_0=\frac{3}{8},\quad \lambda_1=\frac{3}{4},\quad \lambda_2=-\frac{3}{8}$$

这样设计出的差分格式

$$y_{n+1}=\frac{1}{8}(9y_n-y_{n-2})+\frac{3h}{8}(y'_{n+1}+2y'_n-y'_{n-1})$$

称作 **Hamming 格式**. 它是隐式的,有 4 阶精度.

习　题　三

1. 列出求解下列初值问题的 Euler 格式:

(1) $y'=x^2+y^2\ (0\leqslant x\leqslant 0.4)$, $y(0)=1$, 取 $h=0.2$;

(2) $y'=\left(\dfrac{y}{x}\right)^2-\dfrac{y}{x}\ (1\leqslant x\leqslant 1.2)$, $y(1)=1$, 取 $h=0.1$.

2. 用 Euler 格式求解初值问题 $y'=ax+b$, $y(0)=0$:

(1) 试导出近似解 y_n 的显式表达式;

(2) 证明整体截断误差为

$$y(x_n)-y_n=\frac{1}{2}anh^2$$

3. 证明改进的 Euler 格式能准确地求解初值问题 $y'=ax+b$, $y(0)=0$.

4. 设计下列两步格式使其精度尽可能地高:

$$y_{n+1}=ay_n+by_{n-1}+h[cf(x_n,y_n)+df(x_{n-1},y_{n-1})]$$

5. 用梯形格式求解初值问题 $y'+y=0$, $y(0)=1$. 试验证其近似解有显式表达式

$$y_n=\left(\frac{2-h}{2+h}\right)^n$$

并证明当 $h\to 0$ 时 y_n 收敛到原初值问题的精确解 $y=e^{-x}$.

6. 用改进的 Euler 格式求解上述题 5.

7. 试设计差分格式

$$y_{n+1}=y_n+h(ay'_n+by'_{n-1})$$

8. 试设计差分格式

$$y_{n+1}=a(y_n+y_{n-1})+h(by'_n+cy'_{n-1})$$

9. 试设计差分格式

$$y_{n+1}=(1-b)y_n+by_{n-1}+\frac{h}{4}[(b+3)y'_{n+1}+(3b+1)y'_{n-1}]$$

证明当 $b\neq-1$ 时方法为二阶，而当 $b=-1$ 时则为三阶.

10. 试列出求解初值问题

$$\begin{cases} y_1'=a_{11}y_1+a_{12}y_2, & y_1(0)=y_1^0 \\ y_2'=a_{21}y_1+a_{22}y_2, & y_2(0)=y_2^0 \end{cases}$$

的改进的 Euler 格式.

第四章 方程求根

许多数学和物理问题归结为解函数方程 $f(x)=0$. 方程 $f(x)=0$ 的解称作它的**根**. 本章仅限于考察实根.

对于非线性方程,在某个范围内往往有不止一个根,而且根的分布情况可能很复杂. 面对这种情况,通常先将所考察的范围划分成若干子段,然后判断哪些子段内有根. 这项手续称作**根的隔离**.

将所求的根隔离开来以后,再在有根子段内找出满足精度要求的近似根. 为此适当选取有根子段内某一点作为根的初始近似,然后运用迭代方法使之逐步精确化.

本章着重介绍方程求根的迭代法.

4.1 根的搜索

4.1.1 根的逐步搜索

首先进行根的隔离,即在给定区间 $[a,b]$ 内判定根的大致分布. 为此从区间的左端点 $x=a$ 出发,按某个预定的步长 h 一步一步地向右跨,每跨一步进行一次**根的搜索**,即检查每一步的起点 x_0 和终点 x_0+h 的函数值是否同号. 如果发现 $f(x_0)$ 与 $f(x_0+h)$ 非同号,即成立

$$f(x_0)\cdot f(x_0+h)<0$$

那么即可找出一个压缩了的有根区间 $[x_0,x_0+h]$.

例 1 考察方程

$$f(x)=x^3-x-1=0$$

注意到 $f(0)<0$, $f(+\infty)>0$, 知 $f(x)=0$ 至少有一个正的实根.

设从 $x=0$ 出发,取 $h=0.5$ 为步长向右进行根的搜索. 列表记录各个节点上函数值的符号(表 4.1),发现在区间 [1.0,1.5] 内必有实根.

表 4.1

x	0	0.5	1.0	1.5
$f(x)$的符号	−	−	−	+

在具体运用上述逐步搜索方法时,步长 h 的选择是个关键. 很明显,只要步

长 h 充分小，运用这种方法通常可以将所求的根隔离开来，甚至可以获得满足精度要求的近似根．不过当 h 缩小时，所要搜索的步数相应增多，从而使计算量增大．因此，如果精度要求比较高，单用这种逐步搜索方法是不切实际的．

现在运用二分技术加速根的搜索过程．

4.1.2　根的二分搜索

假定函数 $f(x)$ 在 $[a,b]$ 上连续，且 $f(a)\cdot f(b)<0$，根据连续函数的性质，方程 $f(x)=0$ 在 $[a,b]$ 内一定有实根．这里假定它在 $[a,b]$ 内有唯一的单实根 x^*．

考察有根区间 $[a,b]$，取其中点 $x_0=(a+b)/2$ 将它划分为两半，然后进行根的搜索，即检查 $f(x_0)$ 与 $f(a)$ 是否同号：如果确系同号，说明所求的根 x^* 在 x_0 的右侧，这时令 $a_1=x_0,b_1=b$；否则 x^* 必在 x_0 的左侧，这时令 $a_1=a,b_1=x_0$（参看图 4.1）．不管出现哪一种情形，新的有根区间 $[a_1,b_1]$ 的长度仅为 $[a,b]$ 的一半，这是一种规模减半的加工手续．

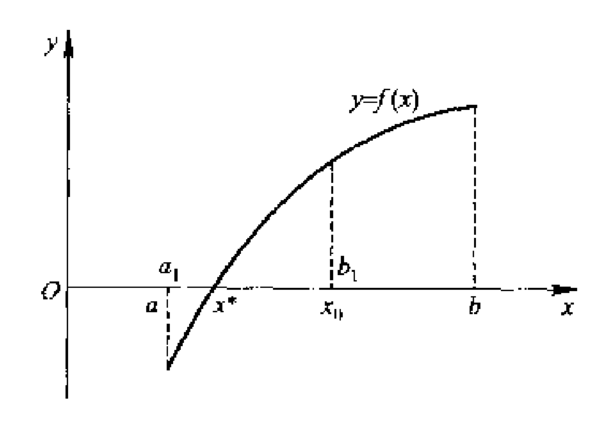


图 4.1　有根区间的二分手续

对压缩了的有根区间 $[a_1,b_1]$ 继续施行上述二分手续，即用中点 $x_1=(a_1+b_1)/2$ 将区间 $[a_1,b_1]$ 再划分为两半，然后判定所求的根 x^* 在 x_1 的哪一侧，从而又确定一个新的有根区间 $[a_2,b_2]$，其长度是 $[a_1,b_1]$ 的一半．

如此反复二分下去，即可得出一系列有根区间

$$[a,b]\supset[a_1,b_1]\supset[a_2,b_2]\supset\cdots$$

其中每个区间长度都是前一区间长度的一半，因此二分 k 次后有根区间 $[a_k,b_k]$ 的长度

$$b_k-a_k=\frac{1}{2^k}(b-a)$$

可见，如果二分过程无限地继续下去，这些有根区间最终必收敛于一点 x^*．该

点显然就是所求的根.

不过在实际计算时,我们不可能完成这个无限过程.其实也没有这种必要,因为实际计算的结果允许带有一定的误差.由于二分 $k+1$ 次后

$$|x^* - x_k| \leqslant \frac{1}{2}(b_k - a_k) = b_{k+1} - a_{k+1}$$

只要有根区间$[a_{k+1}, b_{k+1}]$的长度小于 ε,那么结果 x_k 关于允许误差 ε 就能"准确"地满足方程 $f(x)=0$.

例 2 求方程

$$f(x) = x^3 - x - 1 = 0$$

在区间[1,1.5]内的实根 x^*,要求准确到小数点后的第 2 位.

解 用二分法,这里 $a=1, b=1.5$,且 $f(a)<0$.首先取区间$[a,b]$的中点 $x_0=1.25$ 将区间二等分.由于 $f(x_0)<0$,即 $f(x_0)$与 $f(a)$同号,故所求根必在 x_0的右侧,这时应令 $a_1=x_0=1.25, b_1=b=1.5$,而得到新的有根区间$[a_1,b_1]$(见表 4.2).

对区间$[a_1,b_1]$再用中点 $x_1=1.375$ 二分,并进行根的搜索.二分过程无需赘述.值得指出的是,这时 $f(x_1)$仍然只要同 $f(a)$比较符号,因为 $f(a_1)$与 $f(a)$不可能是异号(按假定$[a,b]$内仅有一根,下面给出的算法也仅仅适用于这种情况).

如此反复二分下去.现在预估所要二分的次数.按误差估计式

$$|x^* - x_k| \leqslant b_{k+1} - a_{k+1} = \frac{1}{2^{k+1}}(b-a)$$

只要二分 6 次,便能达到所要的精度

$$|x^* - x_6| \leqslant 0.005$$

上述二分法的计算结果如表 4.2 所示.

表 4.2

k	a_k	b_k	x_k	$f(x_k)$的符号
0	1	1.5	1.25	-
1	1.25	1.5	1.375	+
2	1.25	1.375	1.312 5	
3	1.312 5	1.375	1.343 8	+
4	1.312 5	1.343 8	1.328 1	+
5	1.312 5	1.328 1	1.320 3	
6	1.320 3	1.328 1	1.324 2	-

算法 4.1(二分法)

步 1　从所给区间$[a,b]$着手二分. 令 $a_1 \Leftarrow a, b_1 \Leftarrow b$.

步 2　取有根区间$[a_1,b_1]$的中点 x 作为近似根.

步 3　通过根的搜索确定二分后新的有根区间$[a_1,b_1]$.

步 4　检查近似根 x 是否满足精度要求;若不满足转步 2 继续二分;若满足精度则输出结果 x 及相应的函数值 $f(x)$.

图 4.2 刻画了二分法的计算流程. 图中,a_1,b_1表示有根区间的左右端点,x 表示近似根.

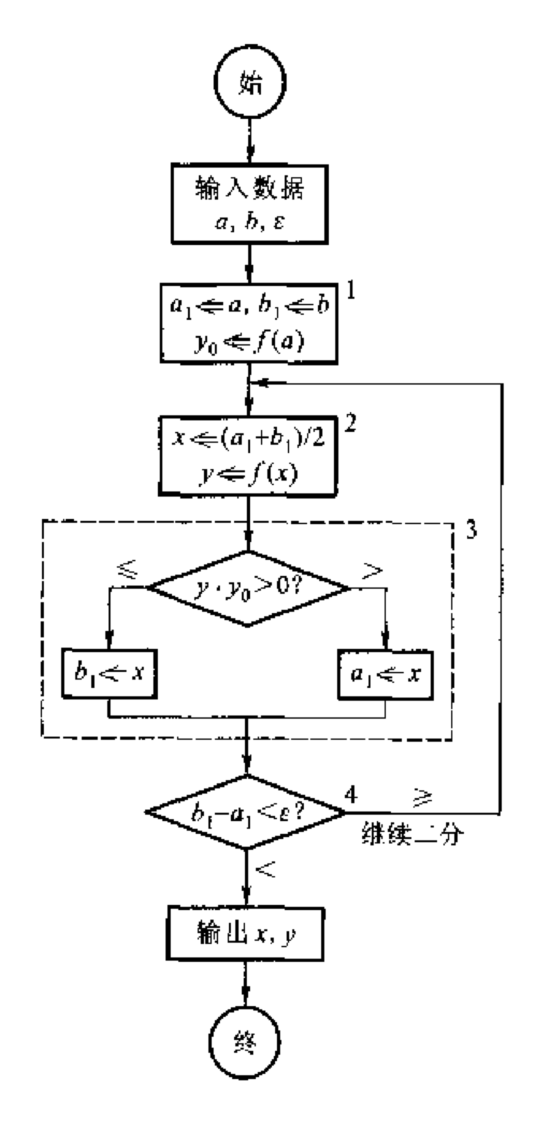


图 4.2　二分法的计算流程

4.2 迭代过程的收敛性

对于复杂方程 $f(x)=0$，具体求根通常分两步走：先用适当方法（譬如根的搜索方法，见 4.1 节）获得根的某个初始近似值 x_0，然后再反复迭代，将 x_0 逐步加工成一系列近似根 $x_1, x_2, \cdots$，直到足够精确为止.

4.2.1 迭代法的设计思想

迭代法是一类重要的逐次逼近方法. 这种方法用某个固定公式反复校正根的近似值，使之逐步精确化，最后得出满足精度要求的结果.

例 3 用迭代法求方程

$$x^3 - x - 1 = 0 \tag{1}$$

在 $x_0 = 1.5$ 附近的一个根.

解 设将方程(1)改写成下列形式

$$x = \sqrt[3]{x+1} \tag{2}$$

用所给初始近似 $x_0 = 1.5$ 代入式(2)的右端，得到

$$x_1 = \sqrt[3]{x_0 + 1} = 1.357\ 21$$

计算结果表明根 x_1 的精度不高. 如果改用 x_1 作为近似值再代入式(2)的右端，又得

$$x_2 = \sqrt[3]{x_1 + 1} = 1.330\ 86$$

由于 x_2 与 x_1 的偏差依然不可忽略，再取 x_2 重复上述步骤. 如此继续下去，这种逐步校正的过程称作迭代过程，这里迭代公式为

$$x_{k+1} = \sqrt[3]{x_k + 1}, \quad k = 0, 1, 2, \cdots \tag{3}$$

表 4.3 记录了各步迭代的计算结果. 可以看到，如果仅取 6 位有效数字，那么迭代值 x_7 与 x_8 相同，这时继续迭代已失去意义，从而得出所求的根为

$$x^* \approx 1.324\ 72$$

表 4.3

k	x_k	k	x_k
0	1.5	5	1.324 76
1	1.357 21	6	1.324 73
2	1.330 86	7	1.324 72
3	1.325 88	8	1.324 72
4	1.324 94		

再考察一般情形.方程 $f(x)=0$ 的求根之所以困难,症结所在是由于它是隐式的.问题在于如何将隐式方程显式化,为此将方程 $f(x)=0$ 改写成

$$x=\varphi(x) \tag{4}$$

的形式,这个方程的左端仅为变元 x,它具有显式的外表,但其右端仍含有未知的 x,因而其本质依然是隐式的.如何将"形显实隐"的形式(4)真正地显式化呢?

显式化是个逐步化归的过程.如果给出根的某个近似值 x_k,将它代入式(4)的右端,则它立即变成显式的:

$$x_{k+1}=\varphi(x_k)$$

这样,从给定的初值 x_0 出发,按

$$x_{k+1}=\varphi(x_k),\quad k=0,1,2,\cdots \tag{5}$$

反复地进行显式计算,即可生成一个序列 $x_1,x_2,\cdots$.如果这个数列有极限,则称迭代过程(5)是**收敛**的,而极限值

$$x^*=\lim_{k\to\infty}x_k$$

显然就是方程 $x=\varphi(x)$ 即 $f(x)=0$ 的根.

这就是求解函数方程的**迭代法**.这种方法依据某个固定公式 $x_{k+1}=\varphi(x_k)$,$k=0,1,2,\cdots$逐步加工所给初值 x_0,结果生成根的近似值序列 $x_1,x_2,\cdots$.这里函数 $\varphi(x)$称**迭代函数**.

迭代法是否有效,关键在于确保其收敛性.问题在于,迭代函数 $\varphi(x)$该怎样设计才能保证迭代过程收敛呢?

4.2.2 线性迭代的启示

为使迭代法有效,必须保证它的收敛性.一个发散(即不收敛)的迭代过程,纵使进行千万步迭代,其结果也是毫无价值的.

这里先考察迭代函数 $\varphi(x)$为线性函数的简单情形,以获得某种直观的启示.

设 $\varphi(x)$是如下形式的线性函数

$$\varphi(x)=Lx+d,\quad L>0$$

即考察如下形式的函数方程

$$x=Lx+d$$

相应的迭代公式具有形式

$$x_{k+1}=Lx_k+d$$

将上面两个式子相减,知

$$x^*-x_{k+1}=L(x^*-x_k)$$

式中 x^* 是所给方程的精确根,因而关于迭代误差 $e_k=|x^*-x_k|$有

$$e_{k+1} = Le_k$$

据此反复递推，有

$$e_k = L^k e_0 \tag{6}$$

由此可见，在线性迭代的情形，为要保证迭代过程收敛，即 $e_k \to 0$，按式(6)只要保证迭代误差 e_k 具有一致的压缩性，即满足条件

$$L < 1$$

4.2.3 压缩映像原理

进而考察一般情形. 设用迭代公式 $x_{k+1} = \varphi(x_k)$ 求方程 $x = \varphi(x)$ 在区间 $[a,b]$ 内的一个根 x^*，依微分中值定理有

$$\begin{aligned} x^* - x_{k+1} &= \varphi(x^*) - \varphi(x_k) \\ &= \varphi'(\xi)(x^* - x_k) \end{aligned} \tag{7}$$

式中 ξ 是 x^* 与 x_k 之间某一点. 由此得知，如果存在定数 L，$0 \leqslant L < 1$，使得对于任意 $x \in [a,b]$ 一致地成立

$$|\varphi'(x)| \leqslant L$$

则据式(7)有

$$|x^* - x_{k+1}| \leqslant L|x^* - x_k| \tag{8}$$

据此反复递推，对迭代误差 $e_k = |x^* - x_k|$ 这里同样有

$$e_k \leqslant L^k e_0$$

由于 $0 \leqslant L < 1$，因而 $e_k \to 0 (k \to \infty)$，故迭代收敛.

需要指出的是，在上述论证过程中应当保证一切迭代值 x_k 全落在区间 $[a,b]$ 内，为此要求对任意 $x \in [a,b]$ 总有

$$\varphi(x) \in [a,b]$$

从而有下述**压缩映像原理**：

定理 1 设 $\varphi(x)$ 在 $[a,b]$ 上具有连续的一阶导数，且满足下列两项条件：

(1) **封闭性条件** 对于任意 $x \in [a,b]$ 总有 $\varphi(x) \in [a,b]$；

(2) **压缩性条件** 存在定数 L，$0 \leqslant L < 1$，使对于任意 $x \in [a,b]$ 一致地成立

$$|\varphi'(x)| \leqslant L \tag{9}$$

则迭代过程 $x_{k+1} = \varphi(x_k)$ 对于任给初值 $x_0 \in [a,b]$ 均收敛于方程 $x = \varphi(x)$ 的根.

4.2.4 局部收敛性

上述压缩映像原理(定理 1)要求迭代函数 $\varphi(x)$ 在某个区间 $[a,b]$ 内一致

地满足压缩性条件(9),这项要求很苛刻,实际应用时很难确定这样的范围$[a,b]$.下面退一步考察迭代过程的局部收敛性.

定义 1　称一种迭代过程在根 x^* **邻近收敛**,如果存在邻域 $\Delta:|x-x^*|\leqslant\delta$ (δ 为某个定数,可以足够小),使迭代过程对任给初值 $x_0\in\Delta$ 均收敛.

这种在根的邻近所具有的收敛性称作**局部收敛性**.局部收敛性要求所选取的初值 x_0 足够准确.

定理 2　设 $\varphi(x)$ 在方程 $x=\varphi(x)$ 的根 x^* 的邻近有连续的一阶导数,且成立

$$|\varphi'(x^*)|<1$$

则迭代过程 $x_{k+1}=\varphi(x_k)$ 在 x^* 邻近具有局部收敛性.

证　由于 $|\varphi'(x^*)|<1$,存在充分小邻域 $\Delta:|x-x^*|\leqslant\delta$,使当 $x\in\Delta$ 时成立

$$|\varphi'(x)|\leqslant L<1$$

这里 L 为某个定数,又易知当 $x\in\Delta$ 时 $\varphi(x)\in\Delta$,故由定理 1 可以断定 $x_{k+1}=\varphi(x_k)$ 对于任意初值 $x_0\in\Delta$ 均收敛.定理得证.

例 4　用迭代法求方程 $x=e^{-x}$ 在 $x_0=0.5$ 附近的一个根 x^*,要求精度为 10^{-5}.

解　这里迭代函数

$$\varphi(x)=e^{-x},\quad \varphi'(x)=-e^{-x}$$

在 $x_0=0.5$ 附近有

$$\varphi'(x^*)\approx\varphi'(x_0)=-e^{-0.5}\approx-0.6$$

因而据定理 2 知,迭代过程 $x_{k+1}=e^{-x_k}$ 具有局部收敛性.

事实上,取初值 $x_0=0.5$,按这一迭代公式迭代 18 次即得满足精度要求的根 0.567 141(见表 4.4).所求根的准确值为 0.567 143.

表 4.4

k	x_k	k	x_k
0	0.500 000	10	0.566 907
1	0.606 531	11	0.567 277
2	0.545 239	12	0.567 067
3	0.579 703	13	0.567 186
4	0.560 065	14	0.567 119
5	0.571 172	15	0.567 157
6	0.564 863	16	0.567 135
7	0.568 438	17	0.567 148
8	0.566 409	18	0.567 141
9	0.567 560		

4.2.5 迭代过程的收敛速度

一种迭代过程要具有实用价值，不但需要肯定它是收敛的，还要求它收敛得比较快.

由考察收敛性定理的定理2，前已指出，当 $|\varphi'(x^*)|<1$ 时迭代过程具有局部收敛性，进一步分析容易看出，值 $|\varphi'(x^*)|$ 越小，误差的压缩性越显著，这时迭代过程收敛得越快.

定义2 对于收敛的迭代过程 $x_{k+1}=\varphi(x_k)$，若 $\varphi'(x^*)\neq 0$ 则称该迭代过程为**线性收敛**，而当 $\varphi'(x^*)=0$ 时称迭代过程具有**平方收敛性**.

具有平方收敛性的迭代法是快速收敛的.

4.3 开 方 法

开方法是古代数学中一颗璀璨的明珠. 无论是古巴比伦数学还是中华传统数学，古先民早已熟练地掌握求开方值的计算方法. 开方算法是迭代法一个生动的范例. 引论0.3节已介绍过开方法，这里进一步剖析开方法的设计机理，目的在于引申出方程求根的一般方法.

4.3.1 开方公式的建立

大家知道，对于给定 $a>0$，求开方值$\sqrt{a}$就是要求解二次方程

$$x^2-a=0 \tag{10}$$

为此，可以运用校正技术设计从预报值 x_k 生成校正值 x_{k+1} 的迭代公式(参看引论0.3节). 自然希望校正值

$$x_{k+1}=x_k+\Delta x$$

能更好满足所给方程(10)：

$$x_k^2+2x_k\Delta x+(\Delta x)^2\approx a$$

这是个关于校正量 Δx 的近似关系式，如果从中删去二次项$(\Delta x)^2$，即可化归为一次方程

$$x_k^2+2x_k\Delta x=a \tag{11}$$

解之有

$$\Delta x=\frac{a-x_k^2}{2x_k}$$

从而关于校正值 $x_{k+1}=x_k+\Delta x$ 有如下**开方公式**

$$x_{k+1}=\frac{1}{2}\left(x_k+\frac{a}{x_k}\right),\quad k=0,1,2,\cdots \tag{12}$$

上述演绎过程表明，开方法(12)的设计思想是逐步线性化，即将二次方程(10)的求解化归为一次方程(11)求解过程的重复.

开方公式(12)规定了预报值 x_k 与校正值 x_{k+1} 之间的一种函数关系 $x_{k+1}=\varphi(x_k)$，这里

$$\varphi(x)=\frac{1}{2}\left(x+\frac{a}{x}\right)$$

为开方法的**迭代函数**.

再考察开方公式(12)的直观含义.

对于开方值 $\sqrt{a}$ 的某个预报值 x_k，设 $x_k\approx\sqrt{a}$，则相应地有

$$\frac{a}{x_k}\approx\sqrt{a}$$

且成立

$$\frac{a}{x_k}-\sqrt{a}=\frac{\sqrt{a}}{x_k}(\sqrt{a}-x_k)$$
$$\approx\sqrt{a}-x_k$$

可见，这时实际上获得了相伴随的一组预报值 x_k 与 $\frac{a}{x_k}$，它们位于 $\sqrt{a}$ 的左、右两侧，并且与 $\sqrt{a}$ 的间距大致相等，由此得知，$\sqrt{a}$ 差不多是它们两者的算术平均：

$$\sqrt{a}\approx\frac{1}{2}\left(x_k+\frac{a}{x_k}\right)$$

因此，直观上看开方公式(12)的结构是合理的.

4.3.2 开方法的收敛性

开方公式的合理性决定了开方过程的收敛性，即迭代误差 $e_k=|x_k-\sqrt{a}|$ 当 $k\to\infty$ 时趋于 0.

现在证明这一事实. 按式(12)有

$$x_{k+1}-\sqrt{a}=\frac{1}{2x_k}(x_k-\sqrt{a})^2$$

同理有

$$x_{k+1}+\sqrt{a}=\frac{1}{2x_k}(x_k+\sqrt{a})^2$$

两式相除有递推公式

$$\frac{x_{k+1}-\sqrt{a}}{x_{k+1}+\sqrt{a}}=\left(\frac{x_k-\sqrt{a}}{x_k+\sqrt{a}}\right)^2$$

反复递推得

$$\frac{x_k-\sqrt{a}}{x_k+\sqrt{a}}=\left(\frac{x_0-\sqrt{a}}{x_0+\sqrt{a}}\right)^{2^k}$$

令

$$q=\left|\frac{x_0-\sqrt{a}}{x_0+\sqrt{a}}\right|$$

则有

$$\frac{x_k-\sqrt{a}}{x_k+\sqrt{a}}=q^{2^k}$$

显然，若 $x_0>0$ 则有 $0<q<1$，这时有

$$x_k=\frac{1+q^{2^k}}{1-q^{2^k}}\sqrt{a}\to\sqrt{a}$$

由此得知

定理 3 开方法(12)对任意给定初值 $x_0>0$ 均收敛.

开方法的初值可以随意选取（关于初值 $x_0>0$ 的要求是不言而喻的），并且收敛速度很快，如此优秀的迭代算法是十分罕见的. 开方法是高效算法的一个生动的范例.

方程求根的核心算法是即将介绍的 Newton 法. 人们将会看到，普适性的 Newton 法与开方法是一脉相承的.



4.4 Newton 法

4.4.1 Newton 公式的导出

考察一般形式的函数方程

$$f(x)=0 \tag{13}$$

首先运用校正技术建立迭代公式. 设已知它的近似根 x_k，则自然要求校正值 $x_{k+1}=x_k+\Delta x$ 能更好地满足所给方程(13)：

$$f(x_k+\Delta x)\approx 0$$

将其左端用其线性主部 $f(x_k)+f'(x_k)\Delta x$ 替代，而令

$$f(x_k)+f'(x_k)\Delta x=0$$

据此定出

$$\Delta x=-\frac{f(x_k)}{f'(x_k)}$$

从而关于校正值 $x_{k+1}=x_k+\Delta x$ 有如下计算公式

$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)} \tag{14}$$

这就是著名的 **Newton 公式**.

由此可见,类同于开方法,Newton 法的设计思想依然是,将非线性方程的求根过程逐步线性化.

Newton 公式(14)决定了预报值 x_k 与校正值 x_{k+1} 之间的一种函数关系 $x_{k+1}=\varphi(x_k)$,这里**迭代函数**为

$$\varphi(x)=x-\frac{f(x)}{f'(x)} \tag{15}$$

4.4.2 Newton 法的收敛性

Newton 法有明显的几何解释.方程 $f(x)=0$ 的根 x^* 在几何上解释为曲线 $y=f(x)$ 与 x 轴交点的横坐标.设 x_k 是根 x^* 的某个近似值,对曲线 $y=f(x)$ 上横坐标为 x_k 的点 P_k 引切线,设该切线与 x 轴的交点的横坐标记为 x_{k+1}(见图 4.3),则这样获得的 x_{k+1} 即为按 Newton 法(14)求得的近似根.由于这种几何背景 Newton 法亦称**切线法**.

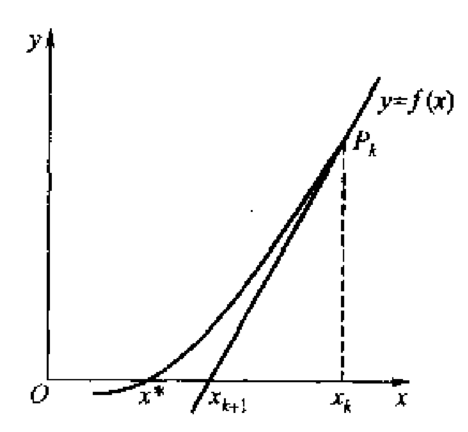


图 4.3 Newton 法的几何背景

考察 Newton 法(14)的收敛速度.利用式(15)求导知

$$\varphi'(x)=\frac{f(x)f''(x)}{[f'(x)]^2}$$

假定 x^* 是方程 $f(x)=0$ 的单根,即 $f(x^*)=0$, $f'(x^*)\neq 0$,则由上式知 $\varphi'(x^*)=0$,因而据定义 2 可以断定:

定理 4 Newton 法(14)在 $f(x)=0$ 的单根 x^* 邻近为平方收敛.

4.4.3 Newton 法的计算流程

Newton 法的突出优点是收敛速度快,且算法的逻辑结构简单.

算法 4.2(Newton 法)

步 1 适当提供迭代初值 x_0.

步 2 按 Newton 公式(14)即

$$x_1 = x_0 - f(x_0)/f'(x_0)$$

求迭代值 x_1.

步 3 检查偏差 $|x_1 - x_0|$ 是否满足精度要求;满足精度则输出结果 x_1,计算结束;否则令 $x_0 \Leftarrow x_1$ 转步 2 继续迭代.

图 4.4 描述了 Newton 法的计算流程. 为保证 Newton 法的计算过程不致中断,自然要求式(14)中的导数值 $f'(x_k)$ 总不为 0;此外,这里设置最大迭代次数 N 以应付迭代过程收敛性差的情形.

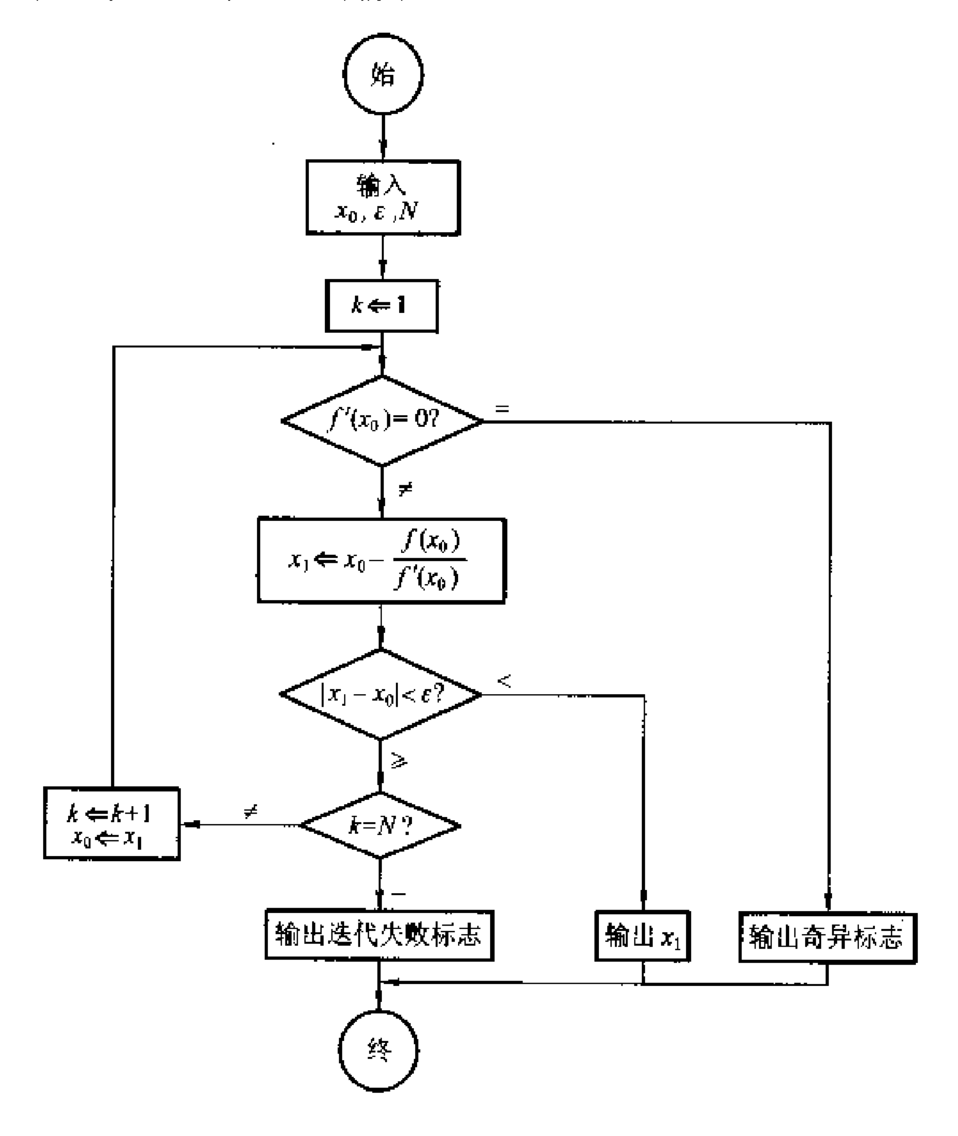


图 4.4 Newton 法的计算流程

例 5　用 Newton 法解方程 $xe^x-1=0$.

解　这里 Newton 公式为

$$x_{k+1}=x_k-\frac{x_k-e^{-x_k}}{1+x_k}.$$

取 $x_0=0.5$，迭代结果如下：

$$x_1=0.571\ 02,\quad x_2=0.567\ 16$$

$$x_3=0.567\ 14,\quad x_4=0.567\ 14$$

这里迭代 3 次得到了精度为 10^{-5} 的结果，可见 Newton 法收敛得很快(试与例 4 比较).

4.4.4　Newton 法应用举例

前述开方法与求倒数值的迭代法(参看引论 0.3 节)是 Newton 法具体应用的两个范例.

大家知道，求开方值 $\sqrt{a}$ 就是要求解方程

$$f(x)=x^2-a=0$$

这时 $f'(x)=2x$，其 Newton 法的迭代函数为

$$\varphi(x)=x-\frac{f(x)}{f'(x)}=\frac{1}{2}\left(x+\frac{a}{x}\right)$$

因而相应的 Newton 公式 $x_{k+1}=\varphi(x_k)$ 就是开方公式(12).

此外，求倒数 $\frac{1}{a}$ 就是要求解方程

$$f(x)=\frac{1}{x}-a=0$$

这里 $f'(x)=-\frac{1}{x^2}$，其 Newton 法的迭代函数为

$$\varphi(x)=x-\frac{f(x)}{f'(x)}=2x-ax^2$$

因而相应的 Newton 公式是

$$x_{k+1}=2x_k-ax_k^2$$

Newton 法有广泛的应用. 为了更为有效地应用 Newton 法，需要进一步将它加以改进和变形.

4.5 Newton 法的改进与变形

4.5.1 Newton 下山法

一般地说，Newton 法的收敛性依赖于初值 x_0 的选取，如果 x_0 偏离 x^* 较远，则 Newton 法可能发散。

例 6 用 Newton 法求方程 $x^3 - x - 1 = 0$ 在 $x = 1.5$ 附近的一个根。

解 取迭代初值 $x_0 = 1.5$，用 Newton 公式

$$x_{k+1} = x_k - \frac{x_k^3 - x_k - 1}{3x_k^2 - 1} \tag{16}$$

计算结果如下

$$x_1 = 1.347\ 83, \quad x_2 = 1.325\ 20, \quad x_3 = 1.324\ 72$$

其中 x_3 的每一位数字都是有效数字。

但是，如果改用 $x_0 = 0.6$ 作为初值，则按式(16)迭代一次得 $x_1 = 17.9$，这个结果反而比 x_0 更偏离了所求的根 x^*。

为了防止迭代发散，通常对迭代过程再附加一项要求，即保证函数值单调下降：

$$|f(x_{k+1})| < |f(x_k)| \tag{17}$$

满足这项要求的算法称**下山法**。

将 Newton 法与下山法结合起来使用，即在下山法保证函数值稳定下降的前提下，用 Newton 法加快收敛速度。为此，将 Newton 法的计算结果

$$\bar{x}_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

与前一步的近似值 x_k 适当加权平均作为新的改进值 x_{k+1}：

$$x_{k+1} = \lambda \bar{x}_{k+1} + (1 - \lambda) x_k$$

或者说，采用下列迭代公式

$$x_{k+1} = x_k - \lambda \frac{f(x_k)}{f'(x_k)} \tag{18}$$

其中，$0 < \lambda \leq 1$ 称**下山因子**。适当选取下山因子 λ，以使单调性条件(17)成立。

下山因子的选择是个逐步探索的过程，从 $\lambda = 1$ 开始反复将因子 λ 的值减半进行试算，一旦单调性条件(17)成立，则称"下山成功"；反之，如果在上述过程中找不到使条件(17)成立的下山因子 λ，则称"下山失败"，这时需另选初值 x_0 重算。

再考察例 6，前面已指出，若取 $x_0=0.6$，则按 Newton 公式(16)求得迭代值 $\bar{x}_1=17.9$，如果取下山因子 $\lambda=\frac{1}{32}$，则由式(18)可求得

$$x_1=\frac{1}{32}\bar{x}_1+\frac{31}{32}x_0=1.140\ 625$$

这个结果纠正了 x_1 的严重偏差.

4.5.2　弦截法

Newton 法的突出优点是收敛速度快，但它还有个明显的缺点：每一步迭代需要提供导数值 $f'(x_k)$. 如果函数 $f(x)$ 比较复杂，致使导数的计算困难，那么使用 Newton 公式是不方便的.

为避开导数的计算，可以改用差商 $\frac{f(x_k)-f(x_0)}{x_k-x_0}$ 替换 Newton 公式(14)中的导数 $f'(x_k)$，即得到下列离散化形式：

$$x_{k+1}=x_k-\frac{f(x_k)}{f(x_k)-f(x_0)}(x_k-x_0) \tag{19}$$

容易看出，这个公式是根据方程 $f(x)=0$ 的等价形式

$$x=x-\frac{f(x)}{f(x)-f(x_0)}(x-x_0) \tag{20}$$

建立的迭代公式.

迭代公式(19)的几何解释如图 4.5 所示，记曲线 $y=f(x)$ 上横坐标为 x_k 的点为 P_k，则差商 $\frac{f(x_k)-f(x_0)}{x_k-x_0}$ 表示弦线 $\overline{P_0P_k}$ 的斜率. 容易看出，按公式(19)求得的 x_{k+1} 实际上是弦线 $\overline{P_0P_k}$ 与 x 轴的交点，因此这种方法称作**弦截法**.

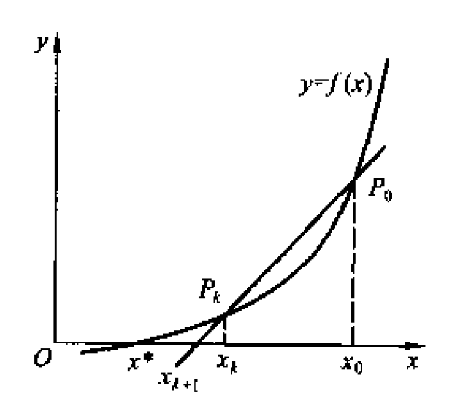


图 4.5　弦截法的几何背景

考察弦截法的收敛性. 直接对式(20)的迭代函数

$$\varphi(x)=x-\frac{f(x)}{f(x)-f(x_0)}(x-x_0)$$

求导得

$$\varphi'(x^*)=1+\frac{f'(x^*)}{f(x_0)}(x^*-x_0)=1-\frac{f'(x^*)}{\dfrac{f(x^*)-f(x_0)}{x^*-x_0}}$$

当 x_0 充分接近 x^* 时 $0<|\varphi'(x^*)|<1$，故由定义 2(见 4.2.5 小节)知弦截法(19)仅为线性收敛.

由此可见，弦截法(19)避开了导数计算，消除了 Newton 法要求提供导数值的困难，但为此在收敛速度方面付出了不可低估的代价.

4.5.3　快速弦截法

为提高弦截法(19)的收敛速度，改用差商$\dfrac{f(x_k)-f(x_{k-1})}{x_k-x_{k-1}}$替代 Newton 公式(14)中的导数 $f'(x_k)$，而导出下列迭代公式：

$$x_{k+1}=x_k-\frac{f(x_k)}{f(x_k)-f(x_{k-1})}(x_k-x_{k-1})$$

这种迭代法称作**快速弦截法**.

快速弦截法虽然提高了收敛速度，但它为此也付出了“沉重”的代价：它在计算 x_{k+1} 时要用到前面两步的信息 x_k,x_{k-1}，即这种迭代法为**两步法**.两步法不能自行启动.使用这种迭代法，在计算前必须先提供两个开始值 x_0 与 x_1.

例 7　用快速弦截法解方程

$$xe^x-1=0$$

解　设取 $x_0=0.5,x_1=0.6$ 作为开始值，用快速弦截法求得的结果如下：

$$x_2=0.567\,54,\quad x_3=0.567\,15,\quad x_4=0.567\,14$$

同例 5 运用 Newton 法的计算结果相比较可以看出，快速弦截法的收敛速度是令人满意的.

小　结

算法设计的理念是将“复杂化归为简单的重复”.方程求根的迭代法突出地表达了这种理念，其设计思想是将隐式的非线性模型逐步地显式化、线性化，从而达到化繁为简的目的.

为保证简单的重复能生成复杂，需要精心设计迭代函数.本书推荐了设计迭代函数的校正技术.校正技术的基础是分析预报值的校正量，通过舍弃高阶小量

的"删繁就简"手续，将难以处理的非线性方程加工成容易求解的线性化的校正方程，然后重复这种加工手续，逐步校正所获得的近似根，直到满足精度要求为止．可见校正技术的设计思想是"删繁就简，逐步求精"．

为要达到"逐步求精"的目的，必须保证迭代过程的收敛性．所谓迭代收敛就是要求迭代误差逐步缩减．如果将迭代误差理解为每一步计算的规模，那么在迭代过程中问题的规模是逐步缩减的，可见迭代法亦可理解为缩减技术的运用，迭代过程又是个"大事化小，小事化了"的过程．

为改善迭代法的有效性，要求尽量提高它的收敛速度．关于迭代加速的研究是极其重要的．为此需要运用松弛技术，将每一迭代步的新值与老值适当加权平均，以得出更高精度的改进值．Newton 下山法表明，这种"优劣互补、化粗为精"的设计策略往往是奏效的．

总而言之，方程求根的迭代法是运用算法设计基本技术——校正技术、缩减技术与松弛技术的又一典范．

题解 4.1 压缩映像原理

提要 运用压缩映像原理考察迭代法的收敛性时，必须全面地考察封闭性与压缩性两个方面．两项条件缺一不可．忽视封闭性条件是运用压缩映像原理解题时常犯的错误之一．

另外，将定理中的压缩性条件 $|\varphi'(x)|\leqslant L<1$ 轻率地替代为 $|\varphi'(x)|<1$，这是运用压缩映像原理解题时另一个常见的原则性错误，请读者留意．

压缩映像原理的反命题是：如果存在定数 $L>1$，使对任给 $x\in[a,b]$ 恒成立

$$|\varphi'(x)|\geqslant L$$

则迭代过程 $x_{k+1}=\varphi(x_k)$ 对于任给初值 $x_0\in[a,b]$ 均发散．

题 1 应用迭代法求解方程

$$x=(\cos x+\sin x)/4$$

并讨论迭代过程的收敛性．

解 这里迭代函数

$$\varphi(x)=(\cos x+\sin x)/4$$

$$\varphi'(x)=(-\sin x+\cos x)/4$$

对于任给实值 x，$\varphi(x)$ 均为实值，且成立

$$|\varphi'(x)|\leqslant\frac{\sqrt{2}}{4}<1$$

因此迭代过程

$$x_{k+1}=(\cos x_k+\sin x_k)/4$$

对于任给实值 x_0 均收敛.

题 2 改写方程 $x^2=2$ 为

$$x=\frac{x}{2}+\frac{1}{x}$$

运用压缩映像原理证明,迭代过程 $x_{k+1}=\frac{x_k}{2}+\frac{1}{x_k}$ 对于任给初值 $x_0>2$ 均收敛于 $\sqrt{2}$.

证 这里迭代函数

$$\varphi(x)=\frac{x}{2}+\frac{1}{x}$$

$$\varphi'(x)=\frac{1}{2}-\frac{1}{x^2}$$

注意到

$$\left(\sqrt{\frac{x}{2}}-\frac{1}{\sqrt{x}}\right)^2\geqslant 0$$

对于任意 $x>0$ 成立

$$\varphi(x)=\frac{x}{2}+\frac{1}{x}\geqslant 2\sqrt{\frac{x}{2}\cdot\frac{1}{\sqrt{x}}}=\sqrt{2}>1$$

而当 $x\geqslant 1$ 时有

$$|\varphi'(x)|\leqslant\frac{1}{2}<1$$

因此,迭代公式 $x_{k+1}=\frac{x_k}{2}+\frac{1}{x_k}$ 对任给初值 $x_0\geqslant 1$ 均收敛于方程 $x^2=2$ 的正根 $\sqrt{2}$.

题 3 基于迭代原理证明

$$\sqrt{2+\sqrt{2+\sqrt{2+\cdots}}}=2$$

证 首先建立迭代公式.令

$$x_k=\underbrace{\sqrt{2+\sqrt{2+\cdots+\sqrt{2}}}}_{k重}$$

则有迭代公式

$$\begin{cases}x_{k+1}=\sqrt{2+x_k}, & k=0,1,2,\cdots\\ x_0=0\end{cases}$$

这里迭代函数

$$\varphi(x)=\sqrt{2+x},\quad \varphi'(x)=\frac{1}{2\sqrt{2+x}}$$

显然当 $x\in[0,2]$ 时 $\varphi(x)\in[0,2]$，且成立

$$|\varphi'(x)|\leqslant\frac{1}{2\sqrt{2}}<1$$

因此这一迭代过程收敛于方程

$$x^2-x-2=0$$

的正根 $x^*=2$.

题 4 基于迭代原理证明

$$\sqrt{1+\sqrt{1+\sqrt{1+\cdots}}}=\frac{1+\sqrt{5}}{2}$$

证 仿照题 3 的证法．这里迭代公式为

$$\begin{cases}x_{k+1}=\sqrt{1+x_k}\\ x_0=1\end{cases}$$

相应的迭代函数是

$$\varphi(x)=\sqrt{1+x}$$

容易验证，当 $x\in[0,2]$ 时 $\varphi(x)\in[0,2]$，且成立

$$|\varphi'(x)|\leqslant\frac{1}{2}<1$$

因此上述迭代过程收敛于方程

$$x^2-x-1=0$$

的正根 $x^*=\dfrac{1+\sqrt{5}}{2}$.

题 5 改写方程 $2^x+x-4=0$ 为 $x=-2^x+4$ 的形式，据此能否用迭代法求所给方程在[1,2]内的实根？

解 这里迭代函数 $\varphi(x)=-2^x+4$，$\varphi'(x)=-2^x\ln 2$，由于当 $x\in[1,2]$ 时

$$|\varphi'(x)|\geqslant 2\ln 2>1$$

故迭代过程 $x_{k+1}=\varphi(x_k)$ 对于任意初值 x_0 均发散.

题 6 改写方程 $2^x+x-4=0$ 为 $x=\ln(4-x)/\ln 2$ 的形式，据此能否用迭代法求所给方程在[1,2]内的实根？

解 这里 $f(x)=2^x+x-4$，注意到 $f(1)<0$，$f(2)>0$，因而方程 $f(x)=0$ 在区间[1,2]内有实根．令

$$\varphi(x)=\ln(4-x)/\ln 2$$

则

$$\varphi'(x)=-\frac{1}{(4-x)\ln 2}$$

注意到当 $x\in[1,2]$ 时 $\varphi(x)\in[1,2]$，且成立

$$|\varphi'(x)|\leqslant\frac{1}{2\ln 2}<1$$

依据压缩映像原理，这里迭代公式 $x_{k+1}=\varphi(x_k)$ 对任意初值 $x_0\in[1,2]$ 均收敛.

题解 4.2　修正的 Newton 法

提要　对于给定方程 $f(x)=0$，Newton 法

$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)},\quad k=0,1,2,\cdots$$

是方程求根的核心算法. 这种方法通常是平方收敛的. 结论是(4.4 节定理 4)：

Newton 法在 $f(x)=0$ 的单根 x^* 邻近为平方收敛.

自然会问，Newton 法在 $f(x)=0$ 的重根邻近收敛性如何？而如果收敛速度不能保证又该怎样处理呢？

题 1　证明解方程 $(x^3-a)^2=0$ 求 $\sqrt[3]{a}$ 的 Newton 法

$$x_{k+1}=\frac{5}{6}x_k+\frac{a}{6x_k^2}$$

仅为线性收敛.

解　注意 $\sqrt[3]{a}$ 是所给方程的二重根，而迭代函数

$$\varphi(x)=\frac{5}{6}x+\frac{a}{6x^2}$$

$$\varphi'(x)=\frac{5}{6}-\frac{a}{3x^3}$$

由于 $\varphi(\sqrt[3]{a})=\sqrt[3]{a}$，且

$$0<\varphi'(\sqrt[3]{a})=\frac{1}{2}<1$$

按 4.2.5 小节定义 2，这一迭代公式在重根 $\sqrt[3]{a}$ 邻近仅为线性收敛.

题 2　证明迭代公式

$$x_{k+1}=\frac{2}{3}x_k+\frac{a}{3x_k^2}$$

是求解方程 $(x^3-a)^2=0$ 的二阶方法，即具有平方收敛性.

解　这里迭代函数

$$\varphi(x)=\frac{2}{3}x+\frac{a}{3x^2},\quad \varphi'(x)=\frac{2}{3}\left(1-\frac{a}{x^3}\right)$$

易知 $\varphi(\sqrt[3]{a})=\sqrt[3]{a}$，且 $\varphi'(\sqrt[3]{a})=0$ 而 $\varphi''(\sqrt[3]{a})\neq0$，故这一迭代法为平方收敛.

题 3 设 x^* 为方程 $f(x)=0$ 的 $m(m\geqslant 2)$ 重根，证明这时 Newton 法

$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)}$$

仅为线性收敛.

证 考察 Newton 法的迭代函数

$$\varphi(x)=x-\frac{f(x)}{f'(x)}$$

$$\varphi'(x)=\frac{f(x)f''(x)}{[f'(x)]^2}$$

由于 x^* 为 $f(x)$ 的 m 重零点，故它具有形式

$$f(x)=(x-x^*)^m g(x),\quad g(x^*)\neq 0$$

这里

$$f'(x)=m(x-x^*)^{m-1}g(x)+(x-x^*)^m g'(x)$$

$$f''(x)=m(m-1)(x-x^*)^{m-2}g(x)+2m(x-x^*)^{m-1}g'(x)+(x-x^*)^m g''(x)$$

于是

$$\varphi'(x)=\frac{g(x)[m(m-1)g(x)+2m(x-x^*)g'(x)+(x-x^*)^2 g''(x)]}{[mg(x)+(x-x^*)g'(x)]^2}$$

从而有

$$\varphi'(x^*)=\frac{m(m-1)[g(x^*)]^2}{[mg(x^*)]^2}=1-\frac{1}{m}$$

因 $m\geqslant 2$，有 $0<\varphi'(x^*)<1$，故这时 Newton 法仅为线性收敛.

题 4 设 x^* 为方程 $f(x)=0$ 的 $m(m\geqslant 2)$ 重根，证明**修正的 Newton 法**

$$x_{k+1}=x_k-m\frac{f(x_k)}{f'(x_k)}$$

具有平方收敛性.

证 这里迭代函数

$$\varphi(x)=x-m\frac{f(x)}{f'(x)}$$

$$\varphi'(x)=1-m+m\frac{f(x)f''(x)}{[f'(x)]^2}$$

利用题 3 的结果得知

$$\varphi'(x^*)=1-m+m\left(1-\frac{1}{m}\right)=0$$

故这一迭代法为平方收敛.

题 5 导出方程 $f(x)=(x^2-a)^2=0$ 的修正的 Newton 法.

解 由于 $\sqrt{a}$ 是所给方程的二重根，这时修正的 Newton 法的迭代函数

$$\varphi(x) = x - 2\frac{f(x)}{f'(x)} = \frac{1}{2}\left(x + \frac{a}{x}\right)$$

相应的迭代公式为

$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{a}{x_k}\right)$$

这是据方程 $x^2 - a = 0$ 求单根$\sqrt{a}$的迭代公式,众所周知它具有平方收敛性.

题 6 导出方程 $f(x) = (x^3 - a)^2 = 0$ 的修正的 Newton 法.

解 类同于题 5,所给方程$(x^3 - a)^2 = 0$ 的修正的 Newton 法正是方程$x^3 - a = 0$ 的 Newton 法

$$x_{k+1} = \frac{2}{3}x_k + \frac{a}{3x_k^2}$$

题 2 直接证明这种方法具有平方收敛性.

习 题 四

1. 利用校正技术设计求$\sqrt[3]{a}$的迭代公式,并证明该迭代过程的收敛性.

2. 早在 1225 年,有人曾求解方程 $x^3 + 2x^2 + 10x - 20 = 0$,并给出了高精度的实根 $x^* = 1.368\ 808\ 107$,试用 Newton 法求得这个结果.

3. 用 Newton 法求下列方程的根,要求计算结果有 4 位有效数字:

(1) $x^3 - 3x - 1 = 0$, $x_0 = 2$

(2) $x^2 - 3x - e^x + 2 = 0$, $x_0 = 1$

4. 对于给定值 $a > 0$,应用 Newton 法导出求$\frac{1}{\sqrt{a}}$而不使用开方运算与除法运算的迭代公式.

5. 考察求解方程 $12 - 3x + 2\cos x = 0$ 的迭代法

$$x_{k+1} = 4 + \frac{2}{3}\cos x_k$$

(1) 证明它对于任意初值 x_0 均收敛;

(2) 证明它具有线性收敛性;

(3) 取 $x_0 = 0.4$,求误差不超过 10^{-3}的近似根.

6. 给出计算

$$x = \cfrac{1}{1 + \cfrac{1}{1 + \cdots}}$$

的迭代公式,讨论迭代过程的收敛性并证明 $x = \frac{\sqrt{5} - 1}{2}$.

7. 求方程 $x^3 - x^2 - 1 = 0$ 在 $x_0 = 1.5$ 附近的一个根,讨论如下几种迭代过程在区间 $[1.3, 1.6]$ 上的敛散性:

(1) 改写方程为 $x^2 = \frac{1}{x-1}$，相应的迭代格式为

$$x_{k+1} = \frac{1}{\sqrt{x_k - 1}}$$

(2) 改写方程为 $x = \frac{1}{x^2} + 1$，相应的迭代格式为

$$x_{k+1} = \frac{1}{x_k^2} + 1$$

(3) 改写方程为 $x^2 = x^3 - 1$，相应的迭代格式为

$$x_{k+1} = \sqrt{x_k^3 - 1}$$

(4) 改写方程为 $x^3 = x^2 + 1$，相应的迭代格式为

$$x_{k+1} = \sqrt[3]{x_k^2 + 1}$$

8. 设方程 $x = \varphi(x)$ 在区间 $[a, b]$ 内有根 x^*，若对 $x \in [a, b]$ 恒成立 $|\varphi'(x)| \geqslant 1$，证明这时迭代过程 $x_{k+1} = \varphi(x_k)$ 对任意初值 $x_0 \in [a, b]$ 均发散.

9. 分别用弦截法和快速弦截法求方程

$$f(x) = x^3 + 2x^2 + 10x - 20 = 0$$

的根，要求精度为 10^{-6}.

10. 设用差商 $\frac{f(x_k + f(x_k)) - f(x_k)}{f(x_k)}$ 替换 Newton 公式中的导数项 $f'(x_k)$，证明这样构造出的迭代公式

$$x_{k+1} = x_k - \frac{[f(x_k)]^2}{f(x_k + f(x_k)) - f(x_k)}$$

在 $f(x) = 0$ 的单根 x^* 附近为二阶收敛.

第五章　线性方程组的迭代法

5.1　引　　言

线性方程组

$$
\begin{cases}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\
\cdots \quad \cdots \quad \cdots \quad \cdots \\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n
\end{cases}
\tag{1}
$$

是个基本的计算模型,它在科学与工程计算中扮演着极其重要的角色.

线性方程组中含有多个变元. 中学数学里人们早已熟知,当变元个数不多时线性方程组的求解是容易的. 问题在于,科学与工程计算中所归结出的线性方程组,其变元个数可能高达几万甚至几百万. 一般形式乃至大规模的线性方程组该如何求解呢?

众所周知,如果系数行列式 $\det(a_{ij})$ 的值异于0,则方程组(1)有唯一解. 然而正如本书开始(参看引论)所指出的,运用Cramer法则求解线性方程组虽然原则上可行,但因其计算量过大而失去实用价值.

求解大规模的线性方程组主要用迭代法. 迭代法的设计思想是将"复杂"化归为"简单"的重复. 这里面对的问题是,相对于形式复杂的式(1),其所对应的"简单"是指什么样的计算模型呢?

5.1.1　变元的相关性

线性方程组(1)可表为

$$\sum_{j=1}^{n} a_{ij}x_j = b_i, \quad i = 1,2,\cdots,n$$

或借助于矩阵的记号

$$
\mathbf{A} = \begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1n} \\
a_{21} & a_{22} & \cdots & a_{2n} \\
\cdots & \cdots & \cdots & \cdots \\
a_{n1} & a_{n2} & \cdots & a_{nn}
\end{bmatrix}
$$

$$b=(b_1,b_2,\cdots,b_n)^{\mathrm{T}},\quad x=(x_1,x_2,\cdots,x_n)^{\mathrm{T}}$$

简洁地表达为

$$Ax=b$$

由此可见,线性方程组(1)本质上是个隐式的计算模型,它的多个变元 x_i 用系数矩阵 **A** 相互"捆绑"在一起.

线性方程组求解的症结所在,是它的各个变元相互关联在一起,或者说,方程组中的各个方程是彼此联立的.如何将方程组中的诸多变元彼此分离开来从而求出它的解呢?

从最简单的做起.

值得注意的是,方程组的解

$$x_i=c_i,\quad i=1,2,\cdots,n$$

可以理解为系数矩阵为单位阵的退化情形.由此可见,线性方程组的求解,就是要设法将其系数矩阵演变成平凡的单位阵.

5.1.2　对角方程组的平凡情形

线性方程组求解的难易程度取决于其系数矩阵的复杂程度.特别地,如果 **A** 是个对角阵

$$A=\begin{bmatrix} a_{11} & & & \mathbf{0} \\ & a_{22} & & \\ & & \ddots & \\ \mathbf{0} & & & a_{nn} \end{bmatrix}$$

这类方程组

$$a_{ii}x_i=b_i,\quad i=1,2,\cdots,n$$

称**对角方程组**,其解为

$$x_i=b_i/a_{ii},\quad i=1,2,\cdots,n$$

对角方程组的各个方程是独立的,并没有真正捆绑在一起,就是说,它只是若干个独立方程的"聚集".对角方程组不能算作严格意义上的"联立方程组".

5.1.3　三角方程组的特殊情形

还有一种容易处理的简单情形.如果其系数矩阵 **A** 是个三角阵,譬如其上三角部分全为零元素:

$$
A = \begin{bmatrix} a_{11} & & & & \mathbf{0} \\ a_{21} & a_{22} & & & \\ a_{31} & a_{32} & a_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix}
$$

对于这种下三角方程组

$$
\begin{cases} a_{11}x_1 = b_1 \\ a_{21}x_1 + a_{22}x_2 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \\ \quad \cdots \cdots \cdots \cdots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \cdots + a_{nn}x_n = b_n \end{cases}
$$

即

$$
\sum_{j=1}^{i} a_{ij}x_j = b_i, \quad i = 1, 2, \cdots, n
$$

只要自上而下逐步回代,即可顺序得出它的解

$$
x_1 \to x_2 \to \cdots \to x_n
$$

这里回代公式为

$$
\begin{cases} x_1 = b_1/a_{11} \\ x_i = \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j\right)/a_{ii}, \quad i = 2, 3, \cdots, n \end{cases}
$$

类似地,对于上三角方程组

$$
\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ \qquad a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \qquad \cdots \cdots \cdots \cdots \\ \qquad\qquad a_{nn}x_n = b_n \end{cases}
$$

即

$$
\sum_{j=i}^{n} a_{ij}x_j = b_i, \quad i = 1, 2, \cdots, n
$$

只要自下而上逐步回代,即可逆序生成它的解

$$
x_n \to x_{n-1} \to \cdots \to x_1
$$

这里回代公式是

$$
\begin{cases} x_n = b_n/a_{nn} \\ x_i = \left(b_i - \sum_{j=i+1}^{n} a_{ij}x_j\right)/a_{ii}, \quad i = n-1, n-2, \cdots, 1 \end{cases}
$$

由此可见，对于三角方程组（无论是下三角方程组还是上三角方程组），只要事先设定诸变元的计算顺序，即可逐步将各个方程分离开来，从而得出它的解.

综上所述，对于系数矩阵为对角阵或三角阵的特殊情形，线性方程组的求解是容易的.

算法设计的机理是将复杂化归为简单的重复. 后文将会看到，求解线性方程组的迭代法，其实质是将所给方程组逐步地对角化或三角化，即将线性方程组的求解过程加工成对角方程组或三角方程组求解过程的重复.

5.2 迭代公式的建立

本节介绍两种基本的迭代法：Jacobi 迭代与 Gauss-Seidel 迭代.

5.2.1 Jacobi 迭代

Jacobi 迭代的设计思想是将所给线性方程组逐步地对角化.

首先考察三阶方程组的具体情形：

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \end{cases} \tag{2}$$

令其左端仅保留对角成分，将其余成分挪到右端，而改写成如下"伪对角形式"：

$$\begin{cases} a_{11}x_1 = b_1 - a_{12}x_2 - a_{13}x_3 \\ a_{22}x_2 = b_2 - a_{21}x_1 - a_{23}x_3 \\ a_{33}x_3 = b_3 - a_{31}x_1 - a_{32}x_2 \end{cases}$$

用**预报值** $x_1^{(k)}, x_2^{(k)}, x_3^{(k)}$ 代入上式右端，求得的解 $x_1^{(k+1)}, x_2^{(k+1)}, x_3^{(k+1)}$ 称**校正值**，这样建立的**预报校正系统**为

$$\begin{cases} a_{11}x_1^{(k+1)} = b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} \\ a_{22}x_2^{(k+1)} = b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)} \\ a_{33}x_3^{(k+1)} = b_3 - a_{31}x_1^{(k)} - a_{32}x_2^{(k)} \end{cases}$$

其求解公式

$$\begin{cases} x_1^{(k+1)} = (b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)})/a_{11} \\ x_2^{(k+1)} = (b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)})/a_{22} \\ x_3^{(k+1)} = (b_3 - a_{31}x_1^{(k)} - a_{32}x_2^{(k)})/a_{33} \end{cases}$$

称作求解方程组(2)的 **Jacobi 迭代公式**.

进而考察一般形式的方程组(1)，从中分离出对角元素，即令其第 i 个方程

的左端仅保留对角元 x_i，而将其余成分挪到右端，则可改写成如下伪对角形式：

$$a_{ii}x_i = b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}x_j, \quad i=1,2,\cdots,n$$

依据这种等价形式可建立迭代法

$$a_{ii}x_i^{(k+1)} = b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}x_j^{(k)}, \quad i=1,2,\cdots,n$$

从而有 **Jacobi 迭代公式**

$$x_i^{(k+1)} = \left(b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij}x_j^{(k)}\right)/a_{ii}, \quad i = 1,2,\cdots,n \tag{3}$$

可以看到，求解线性方程组的 Jacobi 迭代法，其设计思想是将线性方程组的求解归结为对角方程组求解过程的重复.

迭代法因其计算规则简单而易于编写计算程序. 通常用迭代偏差 $\max\limits_{1\le i\le n}|x_i^{(k+1)} - x_i^{(k)}|$ 刻画迭代值的**精度**. 此外，为防止迭代过程不收敛或者收敛速度过于缓慢，我们设置**最大迭代次数** N，如果迭代次数超过 N 尚不能达到精度要求则宣告迭代失败. 下面列出 Jacobi 迭代的计算步骤.

算法 5.1（Jacobi 迭代）

步 1　适当提供迭代初值 $\{x_i^{(0)}\}$.

步 2　按 Jacobi 公式(3)将老值 $\{x_i^{(k)}\}$ 加工成新值 $\{x_i^{(k+1)}\}$.

步 3　若迭代偏差 $e_k = \max\limits_{1\le i\le n}|x_i^{(k+1)} - x_i^{(k)}|$ 小于指定精度 ε，则输出结果，终止计算；否则执行下一步.

步 4　若迭代次数 k 尚未达到最大迭代次数 N，则转步 2 继续迭代；否则输出迭代失败标志，终止计算.

图 5.1 刻画了 Jacobi 迭代的计算流程. 这里设置了两组单元 $\{x_i\}$，$\{y_i\}$ 分别存放每一步迭代的老值与新值. 框 1 的 Jacobi 算式是整个框图的核心.

例 1　用 Jacobi 迭代法解方程组

$$\begin{cases} 10x_1 - x_2 - 2x_3 = 7.2 \\ -x_1 + 10x_2 - 2x_3 = 8.3 \\ -x_1 - x_2 + 5x_3 = 4.2 \end{cases} \tag{4}$$

解　取迭代初值 $x_1^{(0)} = x_2^{(0)} = x_3^{(0)} = 0$ 套用 Jacobi 公式

$$\begin{cases} x_1^{(k+1)} = 0.72 + 0.1x_2^{(k)} + 0.2x_3^{(k)} \\ x_2^{(k+1)} = 0.83 + 0.1x_1^{(k)} + 0.2x_3^{(k)} \\ x_3^{(k+1)} = 0.84 + 0.2x_1^{(k)} + 0.2x_2^{(k)} \end{cases}$$

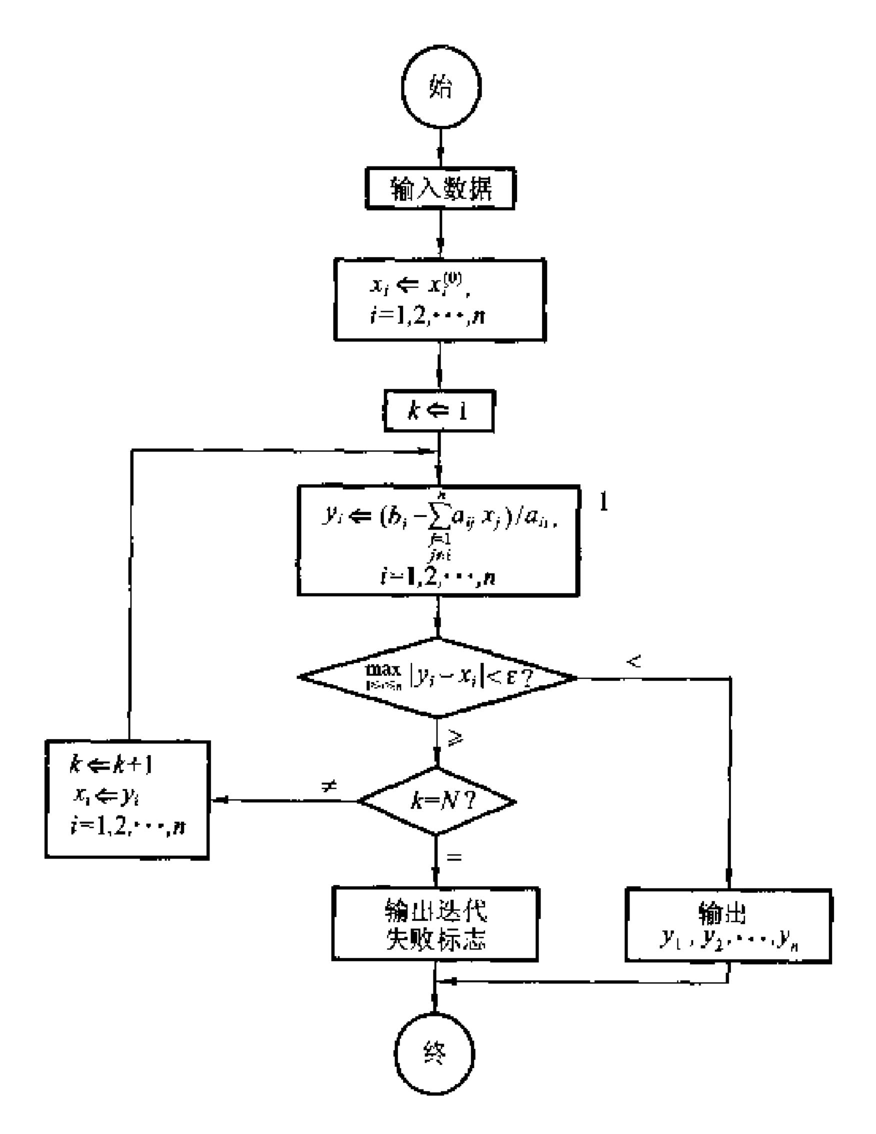


图 5.1　Jacobi 迭代计算流程

反复迭代，计算结果见表 5.1. 可以看到，当迭代次数 k 增大时，迭代值 $x_1^{(k)}$, $x_2^{(k)}$, $x_3^{(k)}$ 会越来越逼近所求的解 $x_1^* = 1.1$, $x_2^* = 1.2$, $x_3^* = 1.3$.

表 5.1

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
0	0.000 00	0.000 00	0.000 00
1	0.720 00	0.083 00	0.840 00
2	0.971 00	1.070 00	1.150 00
3	1.057 00	1.157 10	1.248 20
4	1.085 35	1.185 34	1.282 82
5	1.095 10	1.195 10	1.294 14
6	1.098 34	1.198 34	1.298 04
7	1.099 44	1.199 44	1.299 34
8	1.099 81	1.199 81	1.299 78
9	1.099 94	1.199 94	1.299 92

5.2.2 Gauss-Seidel 迭代

我们再设法将所给方程组逐步三角化,以设计出新的迭代法.

仍然先考察三阶方程组(2),设令其左端仅保留下三角成分,而将其余成分挪到右端,则可改写成如下"伪三角形式":

$$\begin{cases} a_{11}x_1 = b_1 - a_{12}x_2 - a_{13}x_3 \\ a_{21}x_1 + a_{22}x_2 = b_2 - a_{23}x_3 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \end{cases}$$

依据这一等价形式可设计出迭代法

$$\begin{cases} a_{11}x_1^{(k+1)} = b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} \\ a_{21}x_1^{(k+1)} + a_{22}x_2^{(k+1)} = b_2 - a_{23}x_3^{(k)} \\ a_{31}x_1^{(k+1)} + a_{32}x_2^{(k+1)} + a_{33}x_3^{(k+1)} = b_3 \end{cases}$$

它可以看作是关于迭代值 $x_1^{(k+1)}, x_2^{(k+1)}, x_3^{(k+1)}$ 的下三角方程组,用回代法求解,其回代公式

$$\begin{cases} x_1^{(k+1)} = (b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)})/a_{11} \\ x_2^{(k+1)} = (b_2 - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)})/a_{22} \\ x_3^{(k+1)} = (b_3 - a_{31}x_1^{(k+1)} - a_{32}x_2^{(k+1)})/a_{33} \end{cases} \tag{5}$$

称作求解方程组(2)的 **Gauss-Seidel 公式**.

与 Jacobi 公式不同,Gauss-Seidel 公式(5)先设定计算顺序 $x_1^{(k+1)} \to x_2^{(k+1)} \to x_3^{(k+1)}$,然后充分利用新信息进行计算,如用 $x_1^{(k+1)}$ 取代 $x_1^{(k)}$ 计算 $x_2^{(k+1)}$,再用 $x_2^{(k+1)}$ 取代 $x_2^{(k)}$ 计算 $x_3^{(k+1)}$ 等. 由于 Gauss-Seidel 迭代充分利用新信息进行计算,可以预料,它的逼近效果通常比 Jacobi 迭代好.

进而讨论一般形式的方程组(1). 令其左端仅保留下三角成分,将其余成分挪到右端,而加工成如下"伪三角形式"[①]:

$$\sum_{j=1}^{i} a_{ij}x_j = b_i - \sum_{j=i+1}^{n} a_{ij}x_j, \quad i = 1, 2, \cdots, n$$

据此设计出迭代法

$$\sum_{j=1}^{i} a_{ij}x_j^{(k+1)} = b_i - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}$$

---

① 本书约定,和式 $\sum_{j=m}^{l} a_j$ 当 $l < m$ 时其值为 0. 譬如当 $i = n$ 时项 $\sum_{j=i+1}^{n} a_{ij}x_j$,当 $i=1$ 时项 $\sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)}$ 等都是虚设的.

这是关于迭代值 $x_i^{(k+1)}$ 的下三角方程组，自上而下逐步回代即可顺序求出

$$x_1^{(k+1)} \to x_2^{(k+1)} \to \cdots \to x_{i-1}^{(k+1)} \to x_i^{(k+1)} \to \cdots \to x_n^{(k+1)}$$

其求解公式

$$x_i^{(k+1)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)}\right) / a_{ii}, \quad i = 1,2,\cdots,n \tag{6}$$

称为求解方程组(1)的 **Gauss－Seidel 公式**.

Gauss－Seidel 公式(6)的特点是，一旦求出变元 x_i 的新值 $x_i^{(k+1)}$ 以后，老值 $x_i^{(k)}$ 在以后的计算中便失去使用价值，因之可将新值 $x_i^{(k+1)}$ 存放在老值 $x_i^{(k)}$ 所占用的单元内，而将迭代公式(6)表为下列动态形式：

$$x_i \Leftarrow \left(b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{ij} x_j\right) / a_{ii}, \quad i = 1,2,\cdots,n$$

下面列出 Gauss－Seidel 迭代的计算步骤．同 Jacobi 迭代相比较，两种迭代法的计算步骤相类同．不过要特别注意两个迭代公式的差异．

算法 5.2(Gauss－Seidel 迭代)

步 1　适当提供迭代初值 $\{x_i^{(0)}\}$.

步 2　按 Gauss－Seidel 公式(6)将老值 $x_i^{(k)}$ 加工成新值 $x_i^{(k+1)}$.

步 3　若迭代偏差 $e_k = \max\limits_{1\leq i\leq n} \left| x_i^{(k+1)} - x_i^{(k)} \right|$ 小于指定精度 ε，则输出结果，终止计算；否则执行下一步.

步 4　若迭代次数 k 小于事先设定的最大迭代次数 N，则转步 2 继续迭代；否则输出迭代失败标志，终止计算.

图 5.2 描述了 Gauss－Seidel 迭代的计算流程．要注意区分图 5.1 与图 5.2 两者计算框(框 1)的差异.

例 2　用 Gauss－Seidel 迭代求解方程组(4)并与例 1 比较计算结果.

解　这里 Gauss－Seidel 迭代公式为

$$\begin{cases} x_1^{(k+1)} = 0.72 + 0.1x_2^{(k)} + 0.2x_3^{(k)} \\ x_2^{(k+1)} = 0.83 + 0.1x_1^{(k+1)} + 0.2x_3^{(k)} \\ x_3^{(k+1)} = 0.84 + 0.2x_1^{(k+1)} + 0.2x_2^{(k+1)} \end{cases}$$

仍取初值 $x_1^{(0)} = x_2^{(0)} = x_3^{(0)} = 0$ 进行迭代，计算结果见表 5.2．与表 5.1 所列出的 Jacobi 迭代的计算结果相比较可以明显地看出，这里 Gauss－Seidel 迭代的效果比 Jacobi 迭代好.

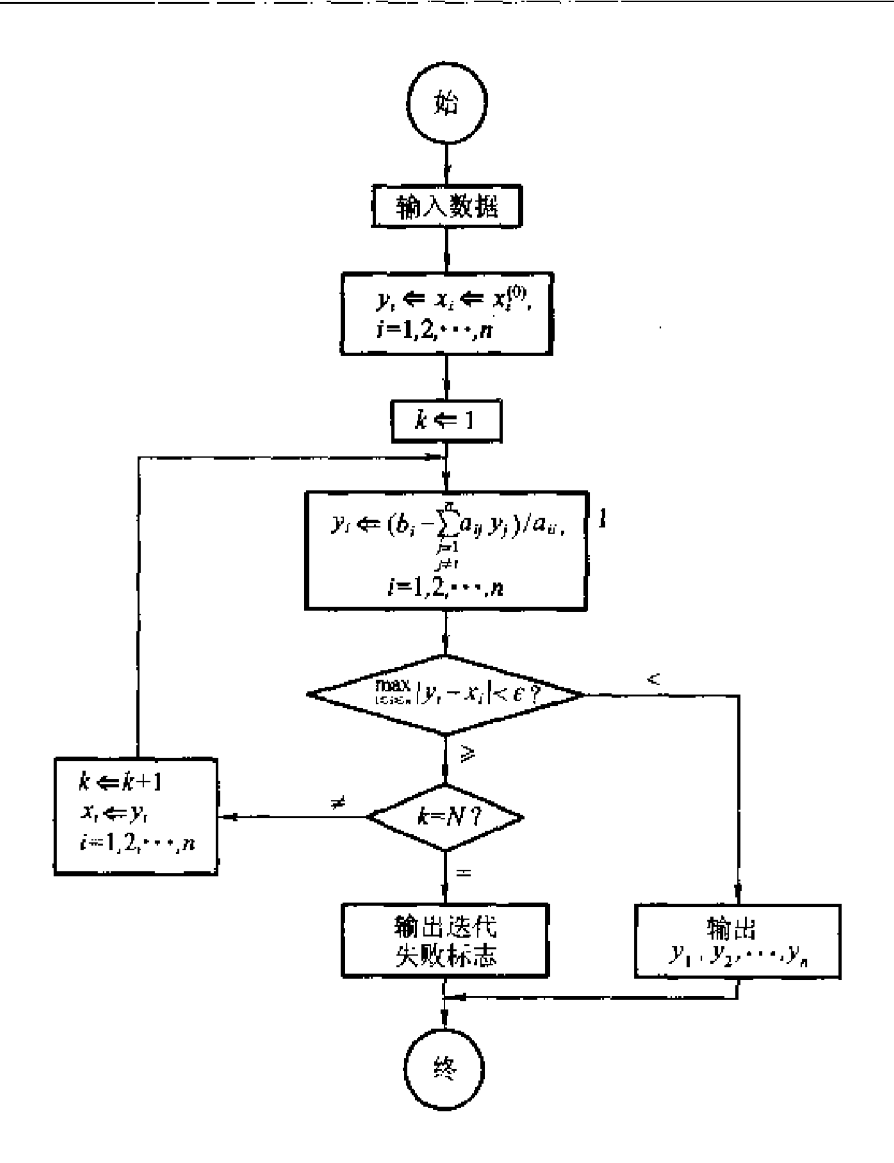


图 5.2 Gauss Seidel 迭代计算流程

表 5.2

k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$
0	0.000 00	0.000 00	0.000 00
1	0.720 00	0.902 00	1.164 40
2	1.043 08	1.167 19	1.282 05
3	1.093 13	1.195 72	1.297 77
4	1.099 13	1.199 47	1.299 72
5	1.099 89	1.199 93	1.299 97
6	1.099 99	1.199 99	1.300 00

以上介绍了求解线性方程组的两种迭代法．由于Gauss－Seidel迭代充分利用了迭代过程中的新信息，一般地说，它的迭代效果要比Jacobi迭代好．但情况并不总是这样，有时Gauss－Seidel迭代比Jacobi迭代收敛得慢；甚至可以举出Jacobi迭代收敛但Gauss－Seidel迭代反而发散的例子．

5.3 迭代过程的收敛性

5.3.1 迭代收敛的概念

为要保证迭代法的有效性，必须要求迭代过程是收敛的．一个发散的迭代过程，即使进行了千万次迭代，其计算结果也是毫无价值的．

例 3 用Jacobi迭代求解方程组

$$\begin{cases} -x_1-x_2+5x_3=4.2 \\ -x_1+10x_2-2x_3=8.3 \\ 10x_1-x_2-2x_3=7.2 \end{cases}$$

解 求解上述方程组的Jacobi迭代公式是

$$\begin{cases} x_1^{(k+1)}=-4.2-x_2^{(k)}+5x_3^{(k)} \\ x_2^{(k+1)}=0.83+0.1x_1^{(k)}+0.2x_3^{(k)} \\ x_3^{(k+1)}=-3.6+5x_1^{(k)}-0.5x_2^{(k)} \end{cases}$$

取迭代初值$(x_1^{(0)},x_2^{(0)},x_3^{(0)})=(0,0,0)$代入上式右端求得

$$x_1^{(1)}=-4.2,\ x_2^{(1)}=0.83,\ x_3^{(1)}=-3.6$$

再迭代一次有

$$x_1^{(2)}=-23.03,\ x_2^{(2)}=-0.31,\ x_3^{(2)}=-25.015$$

如此反复迭代下去，计算结果会越来越偏离所求的解，这个迭代过程是发散的．

需要注意的是，上例所考察的方程组其实同例1的方程组(4)是等价的，两者的差别仅仅是方程的排序不同．这说明，方程组中诸方程的排序方式可能会严重影响迭代过程的收敛性．

在探究收敛性的判别条件之前，首先针对线性方程组进一步明确迭代收敛的概念．

对于一般形式的线性方程组

$$\sum_{j=1}^{n} a_{ij}x_j=b_i,\quad i=1,2,\cdots,n$$

称迭代值序列$(x_1^{(k)},x_2^{(k)},\cdots,x_n^{(k)})$**收敛**到方程组的解$(x_1^*,x_2^*,\cdots,x_n^*)$，如果成立

$$\lim_{k\to\infty} x_i^{(k)} = x_i^*, \quad i = 1,2,\cdots,n$$

按照这个定义,为了判断迭代过程的收敛性,需要检查 n 个数列 $\{x_i^{(k)}\}$ 是否全都收敛. 这就增加了处理的复杂性.

简化分析的一种有效方法是引进**迭代误差**

$$e_k = \max_{1\leq i\leq n} |x_i^{(k)} - x_i^*|$$

基于迭代误差,收敛性的概念可表述为:

定义 1 称迭代序列 $(x_1^{(k)}, x_2^{(k)}, \cdots, x_n^{(k)})$ 收敛到解 $(x_1^*, x_2^*, \cdots, x_n^*)$,如果

$$\lim_{k\to\infty} e_k = 0$$

上一章已看到,迭代过程的收敛性取决于迭代误差的压缩性. 关于线性方程组的迭代法如何保证这种压缩性呢?

5.3.2 二阶方程组的启示

首先考察二阶方程组

$$\begin{cases} a_{11}x_1 + a_{12}x_2 = b_1 \\ a_{21}x_1 + a_{22}x_2 = b_2 \end{cases} \tag{7}$$

其 Jacobi 迭代公式是

$$\begin{cases} x_1^{(k+1)} = -\dfrac{a_{12}}{a_{11}}x_2^{(k)} + \dfrac{b_1}{a_{11}} \\ x_2^{(k+1)} = -\dfrac{a_{21}}{a_{22}}x_1^{(k)} + \dfrac{b_2}{a_{22}} \end{cases} \tag{8}$$

记方程组的精确解为 (x_1^*, x_2^*),则有

$$x_1^{(k+1)} - x_1^* = -\frac{a_{12}}{a_{11}}(x_2^{(k)} - x_2^*)$$

$$x_2^{(k+1)} - x_2^* = -\frac{a_{21}}{a_{22}}(x_1^{(k)} - x_1^*)$$

可见这时迭代误差

$$e_k = \max\{|x_1^{(k)} - x_1^*|, |x_2^{(k)} - x_2^*|\}$$

满足

$$e_{k+1} \leqslant Le_k$$

这里

$$L = \max\left\{\left|\frac{a_{12}}{a_{11}}\right|, \left|\frac{a_{21}}{a_{22}}\right|\right\}$$

由此得知,如果成立

$$|a_{11}| > |a_{12}|, \quad |a_{22}| > |a_{21}| \tag{9}$$

则有

$$0 \leqslant L < 1$$

从而有

$$e_k \leqslant L^k e_0 \to 0$$

即这时 Jacobi 迭代(8)对任给初值均收敛.

再考察求解方程组(7)的 Gauss - Seidel 迭代

$$\begin{cases} x_1^{(k+1)} = -\dfrac{a_{12}}{a_{11}} x_2^{(k)} + \dfrac{b_1}{a_{11}} \\ x_2^{(k+1)} = -\dfrac{a_{21}}{a_{22}} x_1^{(k+1)} + \dfrac{b_2}{a_{22}} \end{cases} \tag{10}$$

这里有

$$x_1^{(k+1)} - x_1^* = -\frac{a_{12}}{a_{11}} (x_2^{(k)} - x_2^*)$$

$$x_2^{(k+1)} - x_2^* = -\frac{a_{21}}{a_{22}} (x_1^{(k+1)} - x_1^*)$$

$$= \frac{a_{21}}{a_{22}} \frac{a_{12}}{a_{11}} (x_2^{(k)} - x_2^*)$$

因之当条件(9)成立时,有

$$|x_1^{(k+1)} - x_1^*| \leqslant L e_k$$

$$|x_2^{(k+1)} - x_2^*| \leqslant L^2 e_k \leqslant L e_k$$

从而仍有误差估计式

$$e_{k+1} \leqslant L e_k$$

可见这时 Gauss - Seidel 迭代(10)对任给初值亦收敛.

后文将要说明上述事实具有普遍意义.

5.3.3 对角占优阵的概念

再剖析收敛性条件(9)的含义. 所给方程组(7)的系数矩阵具有形式

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

条件(9)表明,其对角元素按绝对值大于同行的其他元素. 这种矩阵称作是对角占优的.

进一步推广这个概念.

定义 2 称矩阵 $\mathbf{A} = (a_{ij})_{n \times n}$ 为**对角占优阵**,如果其对角元素 a_{ii} 按绝对值大于同行的其他元素 $a_{ij}(j \neq i)$ 绝对值之和:

$$|a_{ii}| > \sum_{\substack{j=1\\j\neq i}}^{n} |a_{ij}|, \quad i=1,2,\cdots,n$$

即成立

$$L = \max_{1\leqslant i\leqslant n} \sum_{\substack{j=1\\j\neq i}}^{n} \frac{|a_{ij}|}{|a_{ii}|} < 1 \tag{11}$$

系数矩阵为对角占优阵的线性方程组称作是**对角占优**的．实际计算当中归结出来的线性方程组往往具有这种特征，譬如，三次样条插值归结出的基本方程组（参看第一章的式(27)）以及求解常微分方程边值问题列出的差分方程组（参看第三章的式(32)）都是对角占优的．

5.3.4 迭代收敛的一个充分条件

仿照5.3节的做法讨论Jacobi迭代(3)即

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\Big(b_i - \sum_{\substack{j=1\\j\neq i}}^{n} a_{ij}x_j^{(k)}\Big)$$

的收敛性．由于解$\{x_i^*\}$满足

$$x_i^* = \frac{1}{a_{ii}}\Big(b_i - \sum_{\substack{j=1\\j\neq i}}^{n} a_{ij}x_j^*\Big)$$

以上两式相减，有

$$x_i^{(k+1)} - x_i^* = -\frac{1}{a_{ii}} \sum_{\substack{j=1\\j\neq i}}^{n} a_{ij}(x_j^{(k)} - x_j^*)$$

据此得知

$$|x_i^{(k+1)} - x_i^*| \leqslant \sum_{\substack{j=1\\j\neq i}}^{n} \left|\frac{a_{ij}}{a_{ii}}\right| \max_{1\leqslant j\leqslant n} |x_j^{(k)} - x_j^*|$$

从而关于迭代误差 $e_k = \max\limits_{1\leqslant j\leqslant n} |x_j^{(k)} - x_j^*|$ 有估计式

$$e_{k+1} \leqslant \Big(\max_{1\leqslant i\leqslant n} \sum_{\substack{j=1\\j\neq i}}^{n} \left|\frac{a_{ij}}{a_{ii}}\right|\Big) e_k$$

由此可见，如果所给方程组是对角占优的，即式(11)成立，则迭代误差是逐步压缩的．据此断定：

定理1 如果方程组(1)为对角占优，则其Jacobi迭代(3)对于任给初值均收敛．

再考察前面的例1与例3．例1的方程组(4)为对角占优，故其Jacobi迭代是收敛的；与此不同，例3由于方程调序而破坏了系数矩阵的对角占优性，结果导致迭代过程发散．

类似地不难证明：

定理 2　如果方程组(1)是对角占优的，则其 Gauss－Seidel 迭代(6)对任给初值均收敛．

5.4　超松弛迭代

使用迭代法的困难所在是计算量难以估计．有时迭代过程虽然收敛，但由于收敛速度缓慢，使计算量变得很大而失去实用价值．因此，迭代过程的加速具有重要意义．

前已指出，Gauss－Seidel 迭代通常优于 Jacobi 迭代．所谓迭代加速，就是运用松弛技术，将 Gauss－Seidel 迭代值进一步加工成某种松弛值，以尽量改善精度．

再考察简单的二阶方程组

$$\begin{cases} a_{11}x_1 + a_{12}x_2 = b_1 \\ a_{21}x_1 + a_{22}x_2 = b_2 \end{cases}$$

基于 Gauss－Seidel 迭代

$$\begin{cases} x_1^{(k+1)} = -\dfrac{a_{12}}{a_{11}}x_2^{(k)} + \dfrac{b_1}{a_{11}} \\ x_2^{(k+1)} = -\dfrac{a_{21}}{a_{22}}x_1^{(k+1)} + \dfrac{b_2}{a_{22}} \end{cases}$$

的松弛加速公式是

迭代　$\tilde{x}_1^{(k+1)} = -\dfrac{a_{12}}{a_{11}}x_2^{(k)} + \dfrac{b_1}{a_{11}}$

松弛　$x_1^{(k+1)} = \omega\tilde{x}_1^{(k+1)} + (1-\omega)x_1^{(k)}$

迭代　$\tilde{x}_2^{(k+1)} = -\dfrac{a_{21}}{a_{22}}x_1^{(k+1)} + \dfrac{b_2}{a_{22}}$

松弛　$x_2^{(k+1)} = \omega\tilde{x}_2^{(k+1)} + (1-\omega)x_2^{(k)}$

对于一般形式的方程组(1)

$$\sum_{j=1}^{n} a_{ij}x_j = b_i,\quad i = 1,2,\cdots,n$$

其松弛迭代是，对 $i = 1,2,\cdots,n$ 反复执行下列两项计算：

$$\begin{aligned} &\text{迭代}\quad \tilde{x}_i^{(k+1)} = \Big(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\Big)\Big/a_{ii} \\ &\text{松弛}\quad x_i^{(k+1)} = \omega\tilde{x}_i^{(k+1)} + (1-\omega)x_i^{(k)} \end{aligned} \tag{12}$$

很明显，Gauss－Seidel 迭代是松弛因子 $\omega = 1$ 的特殊情形，因而上述松弛迭

代可以看作 Gauss - Seidel 迭代的改进. 由于新值 $\tilde{x}_i^{(k+1)}$ 通常优于老值 $x_i^{(k)}$，在将两者加工成松弛值 $x_i^{(k+1)}$ 时自然要求取松弛因子 $\omega>1$，以尽量发挥新值的优势. 这类松弛迭代(12)称作**超松弛法**. 超松弛迭代简称 **SOR 方法**(Succesive Over - Relaxation).

关于 SOR 方法(12)需要注意两点：一是每一迭代步的迭代值 $\tilde{x}_i^{(k+1)}$ 与松弛值 $x_i^{(k+1)}$ 是交替生成的：

$$\tilde{x}_1^{(k+1)}\to x_1^{(k+1)}\to\tilde{x}_2^{(k+1)}\to x_2^{(k+1)}\to\cdots\to\tilde{x}_n^{(k+1)}\to x_n^{(k+1)}$$

又式(12)，在计算迭代值 $\tilde{x}_i^{(k+1)}$ 时用松弛值 $x_1^{(k+1)},x_2^{(k+1)},\cdots,x_{i-1}^{(k+1)}$ 取代相应的迭代值. 这样，将式(12)迭代与松弛两个环节归并在一起，即得 SOR 方法的下列计算公式：

$$x_i^{(k+1)}=x_i^{(k)}+\frac{\omega}{a_{ii}}\Big(b_i-\sum_{j=1}^{i-1}a_{ij}x_j^{(k+1)}-\sum_{j=i}^{n}a_{ij}x_j^{(k)}\Big),\quad i=1,2,\cdots,n$$

请读者自行绘制 SOR 方法的计算流程图.

超松弛迭代即 SOR 方法具有计算公式简单、编制程序容易等突出优点，它是求解大型稀疏方程组的一种有效方法. 如果松弛因子 ω 选择合适，SOR 方法有可能显著地提高收敛速度.

使用 SOR 方法的关键在于选取合适的松弛因子. 松弛因子的取值对收敛速度影响极大. 实际计算时，通常依据系数矩阵的特点、并结合科学计算的实践经验来选取合适的松弛因子.

例 4 取 $\omega=1.3$，用 SOR 方法解方程组

$$\begin{cases}-4x_1+x_2+x_3+x_4=1\\ x_1-4x_2+x_3+x_4=1\\ x_1+x_2-4x_3+x_4=1\\ x_1+x_2+x_3-4x_4=1\end{cases}$$

要求精度 $\varepsilon=10^{-5}$. 该方程组的精确解为$(-1,-1,-1,-1)^{\mathrm{T}}$.

解 这时 SOR 迭代公式为

$$\begin{cases}x_1^{(k+1)}=x_1^{(k)}-\omega(1+4x_1^{(k)}-x_2^{(k)}-x_3^{(k)}-x_4^{(k)})/4\\ x_2^{(k+1)}=x_2^{(k)}-\omega(1-x_1^{(k+1)}+4x_2^{(k)}-x_3^{(k)}-x_4^{(k)})/4\\ x_3^{(k+1)}=x_3^{(k)}-\omega(1-x_1^{(k+1)}-x_2^{(k+1)}+4x_3^{(k)}-x_4^{(k)})/4\\ x_4^{(k+1)}=x_4^{(k)}-\omega(1-x_1^{(k+1)}-x_2^{(k+1)}-x_3^{(k+1)}+4x_4^{(k)})/4\end{cases}$$

令初值 $x_1^{(0)}=x_2^{(0)}=x_3^{(0)}=x_4^{(0)}=0$，取松弛因子 $\omega=1.3$ 迭代 11 次获得满足精度 $\max\limits_{1\le i\le 4}|x_i^{(k+1)}-x_i^{(k)}|<10^{-5}$ 的结果.

SOR 方法的计算量与松弛因子 ω 的具体选择密切相关. 设用 SOR 方法求

解例4. 下表显示松弛因子 ω 与迭代次数 N 的关系. 表中 $\omega = 1.0$ 为 Gauss-Seidel 迭代，$\omega = 1.3$ 为**最佳松弛因子**.

表 5.3

ω	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8
N	22	17	12	11	14	17	23	33	53

5.5 迭代法的矩阵表示

本章将运用矩阵记号刻画迭代法，以进一步揭示迭代法的实质.

由于线性方程组 $\boldsymbol{Ax} = \boldsymbol{b}$ 是个隐式的计算模型，为要运用迭代法，需要将它改写成"形显实隐"的等价形式

$$x = Gx + d$$

式中 G 称**迭代矩阵**. 据此即可建立迭代公式

$$x^{(k+1)} = Gx^{(k)} + d$$

设计求解方程组 $\boldsymbol{Ax} = \boldsymbol{b}$ 的迭代法，就是要构造出合适的迭代矩阵 G，使得迭代过程 $x^{(k+1)} = Gx^{(k)} + d$ 收敛，并且收敛的速度比较快.

同方程求根的迭代法类似(参看前一章4.1节)，求解线性方程组 $\boldsymbol{Ax} = \boldsymbol{b}$ 的迭代法，其设计思想是将所给计算模型逐步显式化. 问题在于，怎样依据所给系数矩阵 $\boldsymbol{A}$ 设计出合理的迭代矩阵 G 呢?

5.5.1 矩阵的三角分裂

设将方程组 $\boldsymbol{Ax} = \boldsymbol{b}$ 的系数矩阵 $\boldsymbol{A}$ **分裂**成对角阵 $\boldsymbol{D}$，严格下三角阵 $\boldsymbol{L}$ 与严格上三角阵 $\boldsymbol{U}$ 三种成分

$$\boldsymbol{A} = \boldsymbol{D} + \boldsymbol{L} + \boldsymbol{U}$$

即

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \cdots & \cdots & \cdots & & \cdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} =$$

$$\begin{bmatrix} a_{11} & & & & \mathbf{0} \\ & a_{22} & & & \\ & & a_{33} & & \\ & & & \ddots & \\ \mathbf{0} & & & & a_{nn} \end{bmatrix} + \begin{bmatrix} 0 & & & & \mathbf{0} \\ a_{21} & 0 & & & \\ a_{31} & a_{32} & 0 & & \\ \vdots & \vdots & \ddots & \ddots & \\ a_{n1} & a_{n2} & \cdots & a_{n,n-1} & 0 \end{bmatrix} + \begin{bmatrix} 0 & a_{12} & a_{13} & \cdots & a_{1n} \\ & 0 & a_{23} & \cdots & a_{2n} \\ & & \ddots & \ddots & \vdots \\ & & & 0 & a_{n-1,n} \\ \mathbf{0} & & & & 0 \end{bmatrix}$$

则所给方程组 $\boldsymbol{Ax}=\boldsymbol{b}$ 可表为

$$(\boldsymbol{D}+\boldsymbol{L}+\boldsymbol{U})\boldsymbol{x}=\boldsymbol{b}$$

令左端仅保留对角成分，而将其余成分挪到右端，即改写成伪对角形式

$$\boldsymbol{Dx}=-(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{x}+\boldsymbol{b}$$

则有

$$\boldsymbol{x}=-\boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{x}+\boldsymbol{D}^{-1}\boldsymbol{b}$$

据此设计出的迭代法为

$$\boldsymbol{x}^{(k+1)}=-\boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})\boldsymbol{x}^{(k)}+\boldsymbol{D}^{-1}\boldsymbol{b} \tag{13}$$

它是 Jacobi 迭代(3)的矩阵形式，可见 Jacobi 迭代的迭代矩阵为

$$\boldsymbol{G}=-\boldsymbol{D}^{-1}(\boldsymbol{L}+\boldsymbol{U})$$

换一种做法，令方程组 $\boldsymbol{Ax}=\boldsymbol{b}$ 的左端仅保留下三角成分 $\boldsymbol{D}+\boldsymbol{L}$，而改写成下列伪三角形式

$$(\boldsymbol{D}+\boldsymbol{L})\boldsymbol{x}=-\boldsymbol{Ux}+\boldsymbol{b}$$

据此设计出迭代法

$$(\boldsymbol{D}+\boldsymbol{L})\boldsymbol{x}^{(k+1)}=-\boldsymbol{Ux}^{(k)}+\boldsymbol{b} \tag{14}$$

由此得

$$\boldsymbol{Dx}^{(k+1)}=-\boldsymbol{Lx}^{(k+1)}-\boldsymbol{Ux}^{(k)}+\boldsymbol{b}$$

从而有

$$\boldsymbol{x}^{(k+1)}=-\boldsymbol{D}^{-1}(\boldsymbol{Lx}^{(k+1)}+\boldsymbol{Ux}^{(k)})+\boldsymbol{D}^{-1}\boldsymbol{b}$$

容易看出，它是 Gauss－Seidel 公式(6)的矩阵形式，而式(14)表明，Gauss－Seidel 迭代的迭代矩阵为

$$\boldsymbol{G}=-(\boldsymbol{D}+\boldsymbol{L})^{-1}\boldsymbol{U}$$

5.5.2 迭代法的设计机理

进一步运用预报校正技术(参看引论 0.3 节)剖析迭代法的设计机理.

设有解的预报值 $\boldsymbol{x}^{(0)}$，寻求校正值 $\boldsymbol{x}^{(1)}=\boldsymbol{x}^{(0)}+\Delta\boldsymbol{x}$，使之具有更好的精度：

$$\boldsymbol{A}(\boldsymbol{x}^{(0)}+\Delta\boldsymbol{x})\approx\boldsymbol{b}$$

即能较为准确地成立

$$\boldsymbol{A}\Delta\boldsymbol{x}\approx-\boldsymbol{Ax}^{(0)}+\boldsymbol{b}$$

为此，考察如下形式的校正方程

$$\widetilde{A}\Delta x = -Ax^{(0)} + b \tag{15}$$

其中矩阵$\widetilde{A}$称为**校正矩阵**. 自然要求:

(1) $\widetilde{A}$与A相近似,以保证迭代过程收敛.

(2) $\widetilde{A}$的求逆较A方便,以保证迭代公式形式简单.

针对矩阵的三角分裂 $A = D + L + U$,采取下述两种设计策略:

1. 取 A 的对角成分D充当校正矩阵$\widetilde{A}$,这时校正方程(15)具有形式

$$D\Delta x = -Ax^{(0)} + b$$

从而有

$$\begin{aligned} Dx^{(1)} &= D(x^{(0)} + \Delta x) \\ &= (D - A)x^{(0)} + b \\ &= -(L + U)x^{(0)} + b \end{aligned}$$

据此建立的迭代公式

$$x^{(1)} = -D^{-1}(L + U)x^{(0)} + D^{-1}b$$

即 Jacobi 公式(13).

2. 若取 A 的下三角成分 $D + L$ 充当校正矩阵$\widetilde{A}$,这时校正方程(15)具有形式

$$(D + L)\Delta x = -Ax^{(0)} + b$$

从而有

$$\begin{aligned} (D + L)x^{(1)} &= (D + L)(x^{(0)} + \Delta x) \\ &= (D + L - A)x^{(0)} + b \\ &= -Ux^{(0)} + b \end{aligned}$$

据此建立的迭代公式

$$x^{(1)} = -(D + L)^{-1}Ux^{(0)} + (D + L)^{-1}b$$

即 Gauss - Seidal 公式(14).

通过以上分析可以看到,求解线性方程组的 Jacobi 迭代与 Gauss - Seidel 迭代,它们分别基于用对角成分 D 与三角成分 $D + L$ 近似所给矩阵 A.

问题在于,什么样的系数矩阵才具有这种相似性呢?

前面 5.3 节的回答是,对角占优阵正是这种矩阵. 不难想像,对于对角占优阵 A,它的对角成分 D 或三角成分 $D + L$ 可以近似地表达 A,因而这时 Jacobi 迭代与 Gauss - Seidel 迭代是收敛的.

小　　结

本章介绍线性方程组的迭代法. 这里迭代法的设计依然是算法设计基本技术的具体运用.

本章 5.2 节与 5.5 节运用校正技术构造了两种迭代公式——Jacobi 公式与 Gauss－Seidel 公式. 校正技术揭示出迭代公式的合理性(参看 5.5.2 小节),从而为收敛性分析做了铺垫.

收敛性决定了迭代法的有效性. 5.3 节指出,迭代过程是否收敛取决于迭代误差是否具有一致的压缩性. 如果将迭代误差理解为每一迭代步的规模,那么在迭代过程中问题的规模是逐步压缩的.

迭代过程的加速收敛具有重要意义. 5.4 节给出有重要实用价值的 SOR 方法,其关键依然是选取合适的松弛因子.

矩阵是一种强有力的数学工具. 运用矩阵可以对数据体进行整体分析和批量处理.

从矩阵分析的角度来看,系数矩阵为对角阵或三角阵的方程组是简单的,而用迭代法求解线性方程组,其实质是将所给方程组的求解化归为对角方程组或三角方程组求解过程的重复(参看 5.5 节).

将复杂化归为简单的重复,这种做法究竟是否有效,取决于所给系数矩阵的对角成分或三角成分是否同它自身"很相像". 一个很自然的结论是,如果所给系数矩阵是对角占优的,那么它的对角成分或三角成分同它很相像,因而这时迭代法是收敛的(参看 5.3 节).

5.5 节的矩阵分裂技术深刻地揭示了迭代法的本质. 矩阵的三角分裂是平凡的,不需要做任何运算手续,因而,基于矩阵分裂的迭代法为此付出了沉重的代价,迭代过程需要大量的重复计算,甚至可能迭代法不收敛.

值得强调指出的是,比较迭代法的矩阵分裂技术与下一章直接法的矩阵分解技术,它们两者的对立统一性表明算法设计学的深层次的数学美.

题解 5.1　迭代公式的设计

提要　迭代法的设计机理是,将所给方程组 $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$ 的系数矩阵 $\boldsymbol{A}$ 分裂为 $\boldsymbol{A}=\boldsymbol{M}+\boldsymbol{N}$ 的形式,要求其中一个**分裂阵** $\boldsymbol{M}$ 比较容易求逆. 这样,据所给方程组 $(\boldsymbol{M}+\boldsymbol{N})\boldsymbol{x}=\boldsymbol{b}$ 即 $\boldsymbol{M}\boldsymbol{x}=-\boldsymbol{N}\boldsymbol{x}+\boldsymbol{b}$ 可建立起迭代公式

$$\boldsymbol{M}\boldsymbol{x}^{(k+1)}=-\boldsymbol{N}\boldsymbol{x}^{(k)}+\boldsymbol{b}$$

设计迭代法的关键在于选取合适的分裂矩阵 $\boldsymbol{M}$. 设进行矩阵分裂 $\boldsymbol{A}=\boldsymbol{D}+$

$L+U$,其中 D 为对角阵,L 和 U 则分别为严格下三角阵与严格上三角阵,那么,如果取对角阵 D 作为分裂阵 M,则所设计出的迭代法就是 Jacobi 迭代,而若取下三角阵 $D+L$ 作为分裂阵 M,则所设计出的迭代法是 Gauss－Seidel 迭代.

自然会问,针对矩阵分裂 $A=L+D+U$,如果选取上三角阵 $D+U$ 作为分裂阵 M,即采取分裂方式 $A=(D+U)+L$,那么会设计出什么样的迭代法呢?

题 1　考察三阶方程组

$$\begin{cases} a_{11}x_1+a_{12}x_2+a_{13}x_3=b_1 \\ a_{21}x_1+a_{22}x_2+a_{23}x_3=b_2 \\ a_{31}x_1+a_{32}x_2+a_{33}x_3=b_3 \end{cases}$$

试针对计算顺序 $x_3\to x_2\to x_1$ 建立逆序的 Gauss－Seidel 迭代公式.

解　首先改写所给方程组,令其左端仅保留上三角成分(试与本章 5.2 节的做法相比较):

$$\begin{cases} a_{11}x_1+a_{12}x_2+a_{13}x_3=b_1 \\ \qquad\qquad a_{22}x_2+a_{23}x_3=b_2-a_{21}x_1 \\ \qquad\qquad\qquad\qquad a_{33}x_3=b_3-a_{31}x_1-a_{32}x_2 \end{cases}$$

据此可建立预报校正系统

$$\begin{cases} a_{11}x_1^{(k+1)}+a_{12}x_2^{(k+1)}+a_{13}x_3^{(k+1)}=b_1 \\ a_{22}x_2^{(k+1)}+a_{23}x_3^{(k+1)}=b_2-a_{21}x_1^{(k)} \\ a_{33}x_3^{(k+1)}=b_3-a_{31}x_1^{(k)}-a_{32}x_2^{(k)} \end{cases}$$

从而建立起逆序计算 $x_3\to x_2\to x_1$ 的 Gauss－Seidel 迭代公式

$$\begin{cases} x_3^{(k+1)}=(b_3-a_{31}x_1^{(k)}-a_{32}x_2^{(k)})/a_{33} \\ x_2^{(k+1)}=(b_2-a_{21}x_1^{(k)}-a_{23}x_3^{(k+1)})/a_{22} \\ x_1^{(k+1)}=(b_1-a_{12}x_2^{(k+1)}-a_{13}x_3^{(k+1)})/a_{11} \end{cases}$$

题 2　考察矩阵分裂 $A=(D+U)+L$,试给出求解方程组 $Ax=b$ 的迭代公式,并与 Gauss－Seidel 迭代比较计算顺序.

解　据分裂方式 $A=(D+U)+L$ 建立起的迭代公式是

$$(D+U)x^{(k+1)}=-Lx^{(k)}+b$$

即

$$x^{(k+1)}=-D^{-1}(Lx^{(k)}+Ux^{(k+1)})+D^{-1}b$$

其分量形式为

$$x_i^{(k+1)}=\frac{1}{a_{ii}}\Big(b_i-\sum_{j=1}^{i-1}a_{ij}x_j^{(k)}-\sum_{j=i+1}^{n}a_{ij}x_j^{(k+1)}\Big),\quad i=n,n-1,\cdots,1$$

这样设计出的迭代公式,它的每一步逆向计算 $x_n^{(k+1)}\to x_{n-1}^{(k+1)}\to\cdots\to$

$x_1^{(k+1)}$，其计算顺序恰好与 Gauss－Seidel 迭代相反，在这个意义下可以将它看作 Gauss－Seidel 迭代的反方法.

相反相成. 将正反两种 Gauss－Seidel 迭代相匹配，即可生成下述 Gauss－Seidel 预报校正系统.

题 3 令矩阵 $\boldsymbol{A}=(a_{ij})_{n\times n}$ 分裂为 $\boldsymbol{A}=\boldsymbol{L}+\boldsymbol{D}+\boldsymbol{U}$ 的形式，其中 $\boldsymbol{D}$ 为对角阵，$\boldsymbol{L}$ 与 $\boldsymbol{U}$ 分别为严格下三角阵与严格上三角阵，试列出正向反向 Gauss－Seidel 迭代的预报校正系统.

解 其预报校正系统形如

预报 $(\boldsymbol{D}+\boldsymbol{L})\bar{\boldsymbol{x}}^{(k+1)}=-\boldsymbol{U}\boldsymbol{x}^{(k)}+\boldsymbol{b}$

校正 $(\boldsymbol{D}+\boldsymbol{U})\boldsymbol{x}^{(k+1)}=-\boldsymbol{L}\bar{\boldsymbol{x}}^{(k+1)}+\boldsymbol{b}$

相应的分量形式是

$$\bar{x}_i^{(k+1)}=\frac{1}{a_{ii}}\left(b_i-\sum_{j=1}^{i-1}a_{ij}\bar{x}_j^{(k+1)}-\sum_{j=i+1}^{n}a_{ij}x_j^{(k)}\right),\quad i=1,2,\cdots,n$$

$$x_i^{(k+1)}=\frac{1}{a_{ii}}\left(b_i-\sum_{j=1}^{i-1}a_{ij}\bar{x}_j^{(k+1)}-\sum_{j=i+1}^{n}a_{ij}x_j^{(k+1)}\right),\quad i=n,n-1,\cdots,1$$

题解 5.2 迭代过程的收敛性

提要 收敛性的判别很重要. 本书推荐了判别迭代过程收敛的一个充分条件，即检查所给方程组的系数矩阵是否对角占优. 这项条件容易检验，而且，从实际问题中归结出来的大型线性方程组往往具有这种特性.

题 1 设方程组 $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$ 的系数矩阵 $\boldsymbol{A}$ 具有形式：

$$(1)\ \boldsymbol{A}=\begin{bmatrix}1 & a & a\\ a & 1 & a\\ a & a & 1\end{bmatrix}\qquad (2)\ \boldsymbol{A}=\begin{bmatrix}a & 1 & 1\\ 1/a & a & 0\\ 1/a & 0 & a\end{bmatrix}.$$

试问当参数 a 取何值时方能保证矩阵 $\boldsymbol{A}$ 为对角占优.

解 前者要求 $|a|<\frac{1}{2}$，后者要求 $|a|>2$.

题 2 设 $a_{11}a_{22}\neq 0$，证明求解方程组

$$\begin{cases}a_{11}x_1+a_{12}x_2=b_1\\ a_{21}x_1+a_{22}x_2=b_2\end{cases}$$

的 Jacobi 迭代与 Gauss－Seidel 迭代同时收敛或同时发散.

解 先考察 Jacobi 迭代

$$\begin{cases} x_1^{(k+1)} = -\dfrac{a_{12}}{a_{11}} x_2^{(k)} + \dfrac{b_1}{a_{11}} \\ x_2^{(k+1)} = -\dfrac{a_{21}}{a_{22}} x_1^{(k)} + \dfrac{b_2}{a_{22}} \end{cases}$$

这里迭代误差 $e_i^{(k)} = |x_i^{(k)} - x_i^*|, i = 1,2$,满足

$$e_1^{(k+1)} = \left|\frac{a_{12}}{a_{11}}\right| e_2^{(k)}$$

$$e_2^{(k+1)} = \left|\frac{a_{21}}{a_{22}}\right| e_1^{(k)}$$

由此可见,这里保证迭代收敛的充要条件是

$$\left|\frac{a_{12}a_{21}}{a_{11}a_{22}}\right| < 1$$

类似地可以证明,这项条件同样是保证 Gauss - Seidel 迭代收敛的充要条件.

习　题　五

1. 用 Jacobi 迭代与 Gauss - Seidel 迭代求解方程组

$$\begin{cases} 3x_1 + x_2 = 2 \\ x_1 + 2x_2 = 1 \end{cases}$$

要求保留 3 位有效数字.

2. 试列出求解下列方程组的 Jacobi 迭代公式和 Gauss - Seidel 迭代公式,并考察迭代过程的收敛性:

$$\begin{cases} 10x_1 + x_3 - 5x_4 = -7 \\ x_1 + 8x_2 - 3x_3 = 11 \\ 3x_1 + 2x_2 - 8x_3 + x_4 = 23 \\ x_1 - 2x_2 + 2x_3 + 7x_4 = 17 \end{cases}$$

3. 分别用 Jacobi 迭代与 Gauss - Seidel 迭代求解方程组

(1) $\begin{cases} x_1 + 2x_2 = -1 \\ 3x_1 + x_2 = 2 \end{cases}$　　(2) $\begin{cases} x_1 + 5x_2 - 3x_3 = 2 \\ 5x_1 - 2x_2 + x_3 = 4 \\ 2x_1 + x_2 - 5x_3 = -11 \end{cases}$

4. 加工题 3 的方程组,譬如调换方程的排列顺序,以保证迭代过程的收敛性.

5. 将 $\boldsymbol{A}$ 分裂为 $\boldsymbol{A} = \boldsymbol{L} + \boldsymbol{D} + \boldsymbol{U}$,试以 $\boldsymbol{D} + \boldsymbol{U}$ 为校正矩阵设计出一种求解方程组 $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ 的迭代公式.

6. 若 $\boldsymbol{A}$ 可写成如下分块形式

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{bmatrix}$$

其中 $\boldsymbol{A}_{11}$,$\boldsymbol{A}_{22}$均为可逆方阵,且易于求逆,试以

$$\begin{bmatrix} \boldsymbol{A}_{11} & 0 \\ 0 & \boldsymbol{A}_{22} \end{bmatrix}$$

为校正矩阵设计出一种求解方程组 $\boldsymbol{Ax}=\boldsymbol{b}$ 的迭代公式.

7. 分别用 Jacobi 迭代与 Gauss-Seidel 迭代求解方程组

(1) $\begin{cases} x_1+x_3=5 \\ -x_1+x_2=-7 \\ x_1+2x_2-3x_3=-17 \end{cases}$ (2) $\begin{cases} x_1+0.5x_2+0.5x_3=0 \\ 0.5x_1+x_2+0.5x_3=0.5 \\ 0.5x_1+0.5x_2+x_3=-2.5 \end{cases}$

8. 取 $\omega=1.25$ 用松弛法求解下列方程组,要求精度为 $\frac{1}{2}\times10^{-4}$:

$$\begin{cases} 4x_1+3x_2=16 \\ 3x_1+4x_2-x_3=20 \\ -x_2+4x_3=-12 \end{cases}$$

第六章　线性方程组的直接法

上一章介绍了求解线性方程组的迭代法．迭代法的设计思想是，将所给线性方程组的求解过程化归为三角方程组或对角方程组求解过程的重复．

所谓求解线性方程组的直接法，就是通过有限步的运算手续，将所给方程组直接加工成某个三角方程组乃至对角方程组来求解．众所周知的消去法就是这样一类方法，它运用消元手续实现这种加工．

消去法是一种古老的算法．两千年前的中国古代算经《九章算术》中就记载有解线性方程组的消元技术（参看本章6.6节），其设计机理同近代Gauss消去法（参看本章6.5节）一脉相承．

解线性方程组的直接法主要分消去法与矩阵分解方法两大类．为了揭示这两类方法的内在联系，本章6.1节和6.2节首先考察三对角方程组的特殊情形．

解三对角方程组的常用方法是追赶法，追赶法是本章的核心内容．本章介绍的对称方程组（6.4节）乃至一般线性方程组（6.3节与6.5节）的解法，本质上都是追赶法的延伸与拓广．

6.1　追　赶　法

6.1.1　二对角方程组的回代过程

含有大量零元素的矩阵称为**稀疏阵**．对角阵是稀疏阵平凡的特例，其非零元素集中分布在主对角线上，如图6.1所示．

如果矩阵的非零元素集中分布在主对角线以及下（或上）次对角线上，这样的矩阵称作**下（或上）二对角阵**，其结构如图6.2所示，相应的方程组称作**下（或上）二对角方程组**．

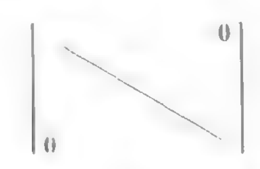


图6.1　对角阵示意图

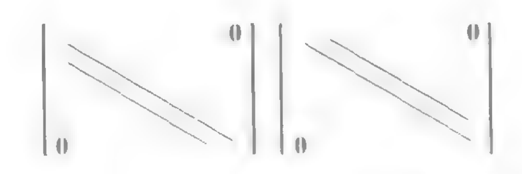


图6.2　下二对角阵与上二对角阵示意图

二对角方程组的求解是容易的．譬如，对于下二对角方程组

$$\begin{cases} b_1x_1 = f_1 \\ a_2x_1 + b_2x_2 = f_2 \\ \cdots\ \cdots\ \cdots\ \cdots \\ a_nx_{n-1} + b_nx_n = f_n \end{cases}$$

即

$$\begin{cases} b_1x_1 = f_1 \\ a_ix_{i-1} + b_ix_i = f_i, \quad i = 2,3,\cdots,n \end{cases}$$

据此自上而下逐步回代即可**顺序**得出它的解

$$x_1 \to x_2 \to \cdots \to x_n$$

这里回代公式为

$$\begin{cases} x_1 = f_1/b_1 \\ x_i = (f_i - a_ix_{i-1})/b_i, \quad i = 2,3,\cdots,n \end{cases}$$

类似地，对于上二对角方程组

$$\begin{cases} b_1x_1 + c_1x_2 = f_1 \\ \cdots\ \cdots\ \cdots\ \cdots \\ b_{n-1}x_{n-1} + c_{n-1}x_n = f_{n-1} \\ b_nx_n = f_n \end{cases}$$

即

$$\begin{cases} b_ix_i + c_ix_{i+1} = f_i, \quad i = 1,2,\cdots,n-1 \\ b_nx_n = f_n \end{cases}$$

据此自下而上逐步回代即可**逆序**得出它的解

$$x_n \to x_{n-1} \to \cdots \to x_1$$

这里回代公式为

$$\begin{cases} x_n = f_n/b_n \\ x_i = (f_i - c_ix_{i+1})/b_i, \quad i = n-1,n-2,\cdots,1 \end{cases}$$

由此可见，对于系数矩阵为二对角阵的简单情形，方程组的求解是容易的．不过需要特别注意解的次序．下二对角方程组的解是顺序得出的，其求解过程称作**追的过程**；反之，上二对角方程组的解则是逆序生成的，其求解过程称作**赶的过程**．一顺一逆，一追一赶．

无论是追的过程还是赶的过程，每做一步，都是将所给下二对角方程组或上二对角方程组化归为变元个数减1的类型相同的二对角方程组，因此，这种回代算法是规模缩减技术的具体应用．

6.1.2 追赶法的设计思想

如果系数矩阵的非零元素集中分布在主对角线及其上、下两条次对角线上(见图 6.3),这类稀疏矩阵称作**三对角阵**.系数矩阵为三对角阵的线性方程组称作是**三对角**的.



图 6.3 三对角阵示意图

前面已指出,作为三对角方程组的特例,二对角方程组的情形是容易处理的. 自然会问,三对角方程组能否化归为二对角方程组来求解呢?

所谓追赶法正是基于这一思想设计出来的.

先考察三阶三对角方程组

$$\begin{cases} b_1x_1 + c_1x_2 = f_1 \\ a_2x_1 + b_2x_2 + c_2x_3 = f_2 \\ a_3x_2 + b_3x_3 = f_3 \end{cases} \tag{1}$$

运用人们所熟知的消元手续进行加工. 第 1 步,将式$(1)_1$中 x_1 的系数化为 1,使之加工成①

$$x_1 + u_1x_2 = y_1 \tag{2}$$

的形式,式中

$$u_1 = c_1/b_1,\ y_1 = f_1/b_1$$

然后利用式(2)从式$(1)_2$中消去 x_1,得

$$a_2(y_1 - u_1x_2) + b_2x_2 + c_2x_3 = f_2$$

再将其中 x_2 的系数化为 1,使之加工成

$$x_2 + u_2x_3 = y_2 \tag{3}$$

的形式,易知

$$u_2 = c_2/(b_2 - a_2u_1)$$
$$y_2 = (f_2 - a_2y_1)/(b_2 - a_2u_1)$$

最后将式(3)代入式$(1)_3$,从中消去 x_2,即可定出

$$x_3 = y_3$$

① 本书以$(\cdot)_i$表示方程组$(\cdot)$的第 i 个方程.

这里

$$y_3=(f_3-a_3y_2)/(b_3-a_3u_2)$$

这样,通过众所周知的消元手续,所给方程组(1)被加工成如下形式的单位上二对角方程组

$$\begin{cases}x_1+u_1x_2=y_1\\x_2+u_2x_3=y_2\\x_3=y_3\end{cases}$$

后者通过回代手续立即解出

$$\begin{cases}x_3=y_3\\x_2=y_2-u_2x_3\\x_1=y_1-u_1x_2\end{cases}$$

6.1.3 追赶法的计算公式

一般地,对于系数阵为三对角阵

$$A=\begin{bmatrix}b_1 & c_1 & & & & \mathbf{0}\\ a_2 & b_2 & c_2 & & & \\ & a_3 & b_3 & c_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & a_{n-1} & b_{n-1} & c_{n-1}\\ \mathbf{0} & & & & a_n & b_n\end{bmatrix}$$

的方程组

$$\begin{cases}b_1x_1+c_1x_2=f_1\\a_ix_{i-1}+b_ix_i+c_ix_{i+1}=f_i, \quad i=2,3,\cdots,n-1\\a_nx_{n-1}+b_nx_n=f_n\end{cases}\tag{4}$$

其加工过程分消元与回代两个环节:

1. **消元过程** 将所给三对角方程组(4)加工成易于求解的单位上二对角方程组

$$\begin{cases}x_i+u_ix_{i+1}=y_i, \quad i=1,2,\cdots,n-1\\x_n=y_n\end{cases}\tag{5}$$

为此所要施行的运算手续是

$$\begin{cases}u_1=c_1/b_1, \ y_1=f_1/b_1\\u_i=c_i/(b_i-a_iu_{i-1}), & i=2,3,\cdots,n-1\\y_i=(f_i-a_iy_{i-1})/(b_i-a_iu_{i-1}), & i=2,3,\cdots,n\end{cases}\tag{6}$$

2. **回代过程** 进一步求解加工得出的二对角方程组(5),其计算公式是

$$\begin{cases} x_n = y_n \\ x_i = y_i - u_i x_{i+1}, \quad i = n-1, n-2, \cdots, 1 \end{cases} \tag{7}$$

显然，上述两个计算环节——无论是消元过程还是回代过程，它们都是规模缩减技术的具体运用．这里可将变元的个数视为线性方程组的规模，这样，每通过消元手续消去一个变元，计算问题的规模便相应地减 1，而直到每个方程仅含一个变元时即可得出所求的解．

需要指出的是，上述消元过程与回代过程这两个环节有着实质性的差异：前者是顺序计算 $y_1 \to y_2 \to \cdots \to y_n$，而后者则是逆序求解 $x_n \to x_{n-1} \to \cdots \to x_1$．如前面 6.1.1 小节所述，通常称前者为**追的过程**，而称后者为**赶的过程**．求解三对角方程组的上述方法则称为**追赶法**．

总之，追赶法的设计机理是将所给三对角方程组(4)化归为简单的二对角方程组(5)来求解，从而达到化繁为简的目的．

6.1.4　追赶法的计算流程

再审视追赶法的计算公式(6)和式(7)，不难看出，这类方法可划分为预处理、追的过程与赶的过程三个环节．

算法 6.1(三对角方程组的追赶法)

1．预处理　生成方程组(5)的系数 u_i 及其除数 d_i．事实上，按式(6)可交替生成 d_i 与 u_i：

$$d_1 \to u_1 \to d_2 \to \cdots \to u_{n-1} \to d_n$$

其计算公式为

$$\begin{cases} d_1 = b_1 \\ u_i = c_i / d_i, \\ d_{i+1} = b_{i+1} - a_{i+1} u_i, \end{cases} \quad i = 1, 2, \cdots, n-1$$

2．追的过程　顺序生成方程组(5)的右端 y_i：

$$y_1 \to y_2 \to \cdots \to y_n$$

据式(6)计算公式为

$$\begin{cases} y_1 = f_1 / d_1 \\ y_i = (f_i - a_i y_{i-1}) / d_i, \quad i = 2, 3, \cdots, n \end{cases}$$

3．赶的过程　逆序得出方程组(5)的解 x_i：

$$x_n \to x_{n-1} \to \cdots \to x_1$$

其计算公式按式(7)为

$$\begin{cases} x_n = y_n \\ x_i = y_i - u_i x_{i+1}, \quad i = n-1, n-2, \cdots, 1 \end{cases}$$

在实际编制程序时，可用数据单元 c_i 和 f_i 分别存放中间结果 u_i 和 y_i，在回代过程中又可将求得的 x_i 再存进单元 f_i 而摈弃其中的老值 y_i. 图 6.4 采用了这种压缩存储方法.

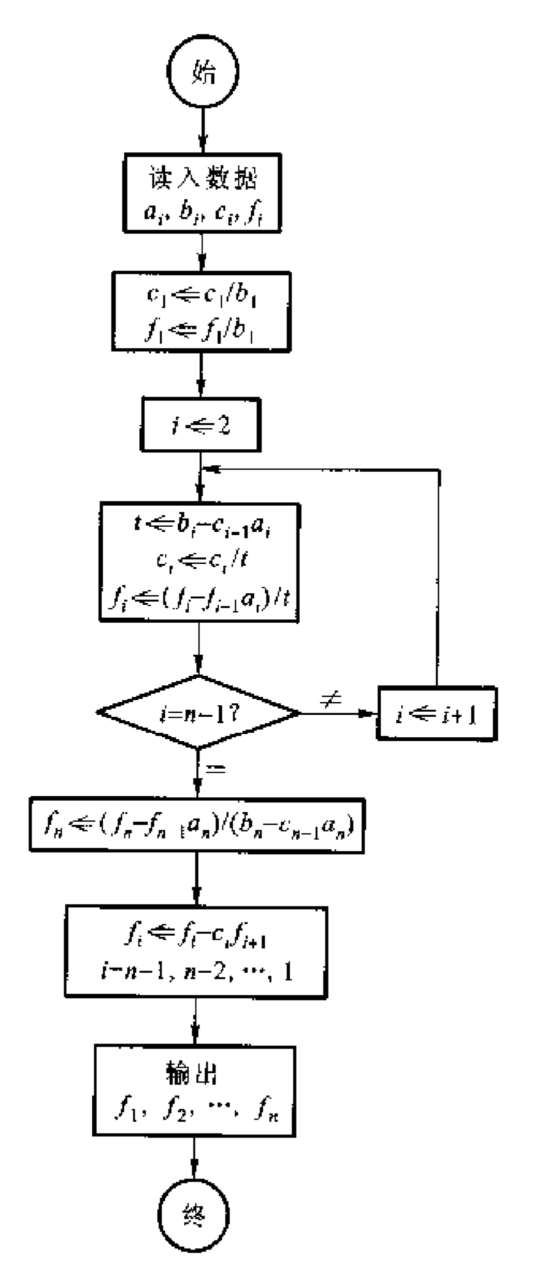


图 6.4 追赶法的计算流程

6.1.5 追赶法的可行性

为使追赶法的计算过程不致中断，必须要求算式中的分母 d_i 全不为 0. 为此考察系数阵为对角占优阵的情形（参看前一章 5.3.3 小节）.

定义 称三对角阵

$$
\mathbf{A}=\begin{bmatrix}
b_1 & c_1 & & & & \mathbf{0} \\
a_2 & b_2 & c_2 & & & \\
 & a_3 & b_3 & c_3 & & \\
 & & \ddots & \ddots & \ddots & \\
 & & & a_{n-1} & b_{n-1} & c_{n-1} \\
\mathbf{0} & & & & a_n & b_n
\end{bmatrix}
$$

为**对角占优阵**，如果其主对角元素的绝对值大于同行次对角元素的绝对值之和，即成立

$$
\begin{aligned}
&|b_1|>|c_1| \\
&|b_i|>|a_i|+|c_i|, \quad i=2,3,\cdots,n-1 \\
&|b_n|>|a_n|
\end{aligned} \tag{8}
$$

定理1 如果所给三对角方程组(4)的系数矩阵是对角占优的，则除数 $d_i(i=1,2,\cdots,n)$ 的值全不为0，从而前述追赶过程不会中断.

证 按对角占优条件(8)

$$|d_1|=|b_1|>|c_1|$$

知 $d_1\neq 0$，又

$$|u_1|=\frac{|c_1|}{|d_1|}=\frac{|c_1|}{|b_1|}<1$$

故再利用(8)有

$$|d_2|=|b_2-a_2u_1|\geqslant|b_2|-|a_2|>|c_2|$$

因而 $d_2\neq 0$. 依此类推知其余 d_i 全不为0，定理得证.

最后统计追赶法的计算量.

追赶法针对三对角方程组的具体特点，在设计算法时将大量的零元素撇开，从而大大地节省了计算量. 易知追赶法大约需要 $3n$ 次加减运算与 $5n$ 次乘除运算.

在计算机上，追赶法是求解三对角方程组的一种有效方法，它具有计算量小、方法简单及算法稳定等优点，因而有广泛的实际应用. 不过，如果三对角方程组的系数矩阵并非对角占优阵，则追赶法可能失效，这时可采用后文6.5节推荐的选主元消去法.

6.2 追赶法的矩阵分解手续

6.2.1 三对角阵的二对角分解

前面已看到，追赶法的设计思想是，通过消元手续将所给三对角方程组(4)

加工成二对角方程组(5),后者求解是容易的. 自然会问,能否运用某种技术,由方程组(4)的系数矩阵 $\boldsymbol{A}$ 直接加工出方程组(5)的系数矩阵

$$\boldsymbol{U}=\begin{bmatrix}1 & u_1 & & & \boldsymbol{0}\\ & 1 & u_2 & & \\ & & \ddots & \ddots & \\ & & & 1 & u_{n-1}\\ \boldsymbol{0} & & & & 1\end{bmatrix}$$

呢?

为了回答这个问题,将所给矩阵 $\boldsymbol{A}$ 分解为上述形式的单位上二对角阵 $\boldsymbol{U}$ 与某个下二对角阵

$$\boldsymbol{L}=\begin{bmatrix}d_1 & & & & \boldsymbol{0}\\ l_2 & d_2 & & & \\ & l_3 & d_3 & & \\ & & \ddots & \ddots & \\ \boldsymbol{0} & & & l_n & d_n\end{bmatrix}$$

的乘积 $\boldsymbol{A}=\boldsymbol{L}\boldsymbol{U}$,即令

$$\begin{bmatrix}b_1 & c_1 & & & \boldsymbol{0}\\ a_2 & b_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-1} & b_{n-1} & c_{n-1}\\ \boldsymbol{0} & & & a_n & b_n\end{bmatrix}=$$

$$\begin{bmatrix}d_1 & & & & \boldsymbol{0}\\ l_2 & d_2 & & & \\ & l_3 & d_3 & & \\ & & \ddots & \ddots & \\ \boldsymbol{0} & & & l_n & d_n\end{bmatrix}\times\begin{bmatrix}1 & u_1 & & & \boldsymbol{0}\\ & 1 & u_2 & & \\ & & \ddots & \ddots & \\ & & & 1 & u_{n-1}\\ \boldsymbol{0} & & & & 1\end{bmatrix}\tag{9}$$

依据上述矩阵展开式,如何利用已给数据 a_i, b_i, c_i 定出分解阵的元素 d_i, l_i 和 u_i 呢?

先考察 $n=4$ 的具体情形,这时矩阵分解(9)表现为

$$\begin{bmatrix}b_1 & c_1 & & \boldsymbol{0}\\ a_2 & b_2 & c_2 & \\ & a_3 & b_3 & c_3\\ \boldsymbol{0} & & a_4 & b_4\end{bmatrix}=\begin{bmatrix}d_1 & & & \boldsymbol{0}\\ l_2 & d_2 & & \\ & l_3 & d_3 & \\ \boldsymbol{0} & & l_4 & d_4\end{bmatrix}\times\begin{bmatrix}1 & u_1 & & \boldsymbol{0}\\ & 1 & u_2 & \\ & & 1 & u_3\\ \boldsymbol{0} & & & 1\end{bmatrix}$$

将这一矩阵关系式按矩阵乘法规则展开,得

$$b_1 = d_1,\quad c_1 = d_1 u_1$$
$$a_2 = l_2,\quad b_2 = l_2 u_1 + d_2,\quad c_2 = d_2 u_2$$
$$a_3 = l_3,\quad b_3 = l_3 u_2 + d_3,\quad c_3 = d_3 u_3$$
$$a_4 = l_4,\quad b_4 = l_4 u_3 + d_4$$

表面上看,这样归结出的关系式是个关于变元 d_i, l_i, u_i 的非线性方程组,它的求解存在实质性的困难. 其实,只要合理地设定计算顺序,解出上述方程组并不困难.

首先注意一个明显的事实:这里矩阵 $\boldsymbol{L}$ 的次对角元素与所给矩阵 $\boldsymbol{A}$ 相同,即成立

$$l_2 = a_2,\ l_3 = a_3,\ l_4 = a_4$$

进一步深入观察不难发现,分解阵 $\boldsymbol{L}, \boldsymbol{U}$ 的其余元素可以逐行依次求出:

$$d_1 = b_1,\ u_1 = c_1/d_1$$
$$d_2 = b_2 - a_2 u_1,\ u_2 = c_2/d_2$$
$$d_3 = b_3 - a_3 u_2,\ u_3 = c_3/d_3$$
$$d_4 = b_4 - a_4 u_3$$

上述分解手续可推广到 n 阶三对角阵的一般情形.

将矩阵关系式(9)按矩阵乘法规则展开,易知分解阵 $\boldsymbol{L}$ 与原矩阵 $\boldsymbol{A}$ 的下次对角线相同,即有 $l_i = a_i$, $i = 2,3,\cdots,n$,从而 $\boldsymbol{L}$ 具有形式

$$\boldsymbol{L} = \begin{bmatrix} d_1 & & & & \mathbf{0} \\ a_2 & d_2 & & & \\ & a_3 & d_3 & & \\ & & \ddots & \ddots & \\ \mathbf{0} & & & a_n & d_n \end{bmatrix}$$

此外依矩阵乘法规则可列出方程

$$\begin{aligned} & b_1 = d_1 \\ & c_i = u_i d_i, \\ & b_{i+1} = u_i a_{i+1} + d_{i+1}, \end{aligned} \qquad i = 1,2,\cdots,n-1$$

据此可逐行定出矩阵 $\boldsymbol{L}$ 与 $\boldsymbol{U}$ 的各个元素,其计算公式为

$$\begin{cases} d_1 = b_1 \\ u_i = c_i/d_i, \\ d_{i+1} = b_{i+1} - u_i a_{i+1}, \end{cases} \qquad i = 1,2,\cdots,n-1 \tag{10}$$

6.2.2 基于矩阵分解的追赶法

基于三对角阵 $\boldsymbol{A}$ 的二对角分解 $\boldsymbol{A}=\boldsymbol{LU}$，所给方程组 $\boldsymbol{Ax}=\boldsymbol{f}$ 即

$$\boldsymbol{L}(\boldsymbol{Ux})=\boldsymbol{f}$$

可化归为 $\boldsymbol{Ly}=\boldsymbol{f}$ 与 $\boldsymbol{Ux}=\boldsymbol{y}$ 两个方程组来求解．前者 $\boldsymbol{Ly}=\boldsymbol{f}$ 是下二对角方程组，其具体形式是

$$\begin{cases} d_1 y_1 = f_1 \\ a_i y_{i-1} + d_i y_i = f_i, \quad i = 2,3,\cdots,n \end{cases}$$

回代解得

$$\begin{cases} y_1 = f_1 / d_1 \\ y_i = (f_i - a_i y_{i-1}) / d_i, \quad i = 2,3,\cdots,n \end{cases} \tag{11}$$

而后者 $\boldsymbol{Ux}=\boldsymbol{y}$ 即前述方程组(5)，其求解公式已由式(7)给出．

这样，在预先进行矩阵分解的前提下，所给三对角方程组(4)可化归为两个二对角方程组来解．这一解题过程可划分为如下三个环节：

1．**预处理** 分解矩阵 $\boldsymbol{A}=\boldsymbol{LU}$，即依式(10)逐行交替计算分解阵 $\boldsymbol{L}$ 与 $\boldsymbol{U}$ 的元素

$$d_1 \to u_1 \to d_2 \to u_2 \to \cdots \to d_{n-1} \to u_{n-1} \to d_n$$

2．**追的过程** 解二对角方程组 $\boldsymbol{Ly}=\boldsymbol{f}$，即依式(11)顺序计算

$$y_1 \to y_2 \to \cdots \to y_n$$

3．**赶的过程** 解单位上二对角方程组 $\boldsymbol{Ux}=\boldsymbol{y}$，即依式(7)逆序求解

$$x_n \to x_{n-1} \to \cdots \to x_1$$

容易看出，上述矩阵分解方法与前述追赶法是一致的．它表明，追赶法的"追—赶"两个过程，其实质是，将所给三对角方程组化归为下二对角方程组与上二对角方程组来求解．

综上所述，矩阵分解 $\boldsymbol{A}=\boldsymbol{LU}$ 是一种代数化方法，分解矩阵 $\boldsymbol{L}$，$\boldsymbol{U}$ 中的元素作为待定参数，它们满足某个代数方程组．值得注意的是，尽管这个方程组是非线性的，但适当设定计算顺序即可归纳出显式化的分解公式．

这类矩阵分解方法亦可用来求解一般形式的**线性**方程组．

6.3 矩阵分解方法

6.3.1 矩阵的 LU 分解

上一节处理三对角方程组的矩阵分解方法，其设计思想对于一般形式的线

性方程组 $\boldsymbol{Ax}=\boldsymbol{b}$ 同样是有效的.

事实上,设系数矩阵 $\boldsymbol{A}$ 可分解成下三角阵 $\boldsymbol{L}$ 与上三角阵 $\boldsymbol{U}$ 的乘积

$$\boldsymbol{A}=\boldsymbol{LU}$$

则所给方程组 $\boldsymbol{Ax}=\boldsymbol{b}$ 即

$$\boldsymbol{L}(\boldsymbol{Ux})=\boldsymbol{b}$$

可化归为两个三角方程组

$$\boldsymbol{Ly}=\boldsymbol{b}$$
$$\boldsymbol{Ux}=\boldsymbol{y}$$

来求解. 正如上一章 5.1 节所指出的,三角方程组有简单的回代公式,求解是方便的.

值得注意的是,类同于三对角方程组的情形,这里,下三角方程组 $\boldsymbol{Ly}=\boldsymbol{b}$ 的回代过程是个**顺序**计算 $y_1\to y_2\to\cdots\to y_n$ 的**追的过程**,而上三角方程组 $\boldsymbol{Ux}=\boldsymbol{y}$ 的回代过程则是**逆序**求解 $x_n\to x_{n-1}\to\cdots\to x_1$ 的**赶的过程**. 因此,上述矩阵分解方法可理解为广义的追赶法.

在具体实施广义追赶法时,将会看到,矩阵的三角分解有下列两种方案可供选择.

6.3.2 矩阵的 Crout 分解

首先考察矩阵分解 $\boldsymbol{A}=\boldsymbol{LU}_1$,这里 $\boldsymbol{L}$ 为下三角阵,$\boldsymbol{U}_1$ 为单位上三角阵,如对三阶矩阵 $\boldsymbol{A}$ 有

$$\begin{bmatrix}a_{11} & a_{12} & a_{13}\\ a_{21} & a_{22} & a_{23}\\ a_{31} & a_{32} & a_{33}\end{bmatrix}=\begin{bmatrix}l_{11} & & \boldsymbol{0}\\ l_{21} & l_{22} & \\ l_{31} & l_{32} & l_{33}\end{bmatrix}\begin{bmatrix}1 & u_{12} & u_{13}\\ & 1 & u_{23}\\ \boldsymbol{0} & & 1\end{bmatrix}$$

按矩阵乘法规则展开,有

$$\begin{aligned}
&a_{11}=\underset{\sim}{l_{11}}, \quad a_{12}=l_{11}\underset{\sim}{u_{12}}, \quad a_{13}=l_{11}\underset{\sim}{u_{13}}\\
&a_{21}=\underset{\sim}{l_{21}}, \quad a_{22}=l_{21}u_{12}+\underset{\sim}{l_{22}}, \quad a_{23}=l_{21}u_{13}+l_{22}\underset{\sim}{u_{23}}\\
&a_{31}=\underset{\sim}{l_{31}}, \quad a_{32}=l_{31}u_{12}+\underset{\sim}{l_{32}}, \quad a_{33}=l_{31}u_{13}+l_{32}u_{23}+\underset{\sim}{l_{33}}
\end{aligned}\tag{12}$$

这样归结出的分解公式是个关于变元 l_{ij}, u_{ij} 的非线性方程组.

为要求解线性方程组,运用矩阵分解方法所归结出的竟然是个非线性方程组,这样处理合适吗?

其实这种疑虑是多余的. 事实上,如果对分解公式(12)设定计算顺序,譬如**逐行**生成分解阵 $\boldsymbol{L}, \boldsymbol{U}_1$ 各个元素(波纹线所示)

$$\begin{aligned}&l_{11}\to u_{12}, u_{13}\\ &\to l_{21}, l_{22}\to u_{23}\\ &\to l_{31}, l_{32}, l_{33}\end{aligned}$$

那么,它的每一步计算都是显式的:

$$l_{11}=a_{11},\quad u_{12}=a_{12}/l_{11},\qquad u_{13}=a_{13}/l_{11}$$
$$l_{21}=a_{21},\quad l_{22}=a_{22}-l_{21}u_{12},\quad u_{23}=(a_{23}-l_{21}u_{13})/l_{22}$$
$$l_{31}=a_{31},\quad l_{32}=a_{32}-l_{31}u_{12},\quad l_{33}=a_{33}-l_{31}u_{13}-l_{32}u_{23}$$

这一事实具有普遍意义,对于一般形式的矩阵分解 $\boldsymbol{A}=\boldsymbol{L}\boldsymbol{U}_1$,这里 $\boldsymbol{L}$ 为下三角阵而 $\boldsymbol{U}_1$ 为单位上三角阵,则所给方程组 $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$,即 $\boldsymbol{L}(\boldsymbol{U}_1\boldsymbol{x})=\boldsymbol{b}$ 可化归为下三角方程组 $\boldsymbol{L}\boldsymbol{y}=\boldsymbol{b}$ 和单位上三角方程组 $\boldsymbol{U}_1\boldsymbol{x}=\boldsymbol{y}$ 来求解. 分解方式 $\boldsymbol{A}=\boldsymbol{L}\boldsymbol{U}_1$ 称作矩阵 $\boldsymbol{A}$ 的 **Crout 分解**.

基于矩阵 $\boldsymbol{A}$ 的 Crout 分解,方程组 $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$ 即

$$\sum_{j=1}^{n}a_{ij}x_j=b_i,\quad i=1,2,\cdots,n$$

的求解分为三个环节.

算法 6.2[①] (Crout 矩阵分解方法)

1. 预处理 实现矩阵分解 $\boldsymbol{A}=\boldsymbol{L}\boldsymbol{U}_1$:对 $i=1,2,\cdots,n$ 计算

$$l_{ij}=a_{ij}-\sum_{k=1}^{j-1}l_{ik}u_{kj},\quad j=1,2,\cdots,i$$

$$u_{ij}=\left(a_{ij}-\sum_{k=1}^{i-1}l_{ik}u_{kj}\right)/l_{ii},\quad j=i+1,i+2,\cdots,n$$

2. 追的过程 解下三角方程组 $\boldsymbol{L}\boldsymbol{y}=\boldsymbol{b}$ 即

$$\sum_{j=1}^{i}l_{ij}y_j=b_i,\quad i=1,2,\cdots,n$$

回代公式为

$$y_i=\left(b_i-\sum_{j=1}^{i-1}l_{ij}y_j\right)/l_{ii},\quad i=1,2,\cdots,n$$

3. 赶的过程 解单位上三角方程组 $\boldsymbol{U}_1\boldsymbol{x}=\boldsymbol{y}$ 即

$$x_i+\sum_{j=i+1}^{n}u_{ij}x_j=y_i,\quad i=1,2,\cdots,n$$

回代公式为

$$x_i=y_i-\sum_{j=i+1}^{n}u_{ij}x_j,\quad i=n,n-1,\cdots,1$$

① 本章约定,和式 $\sum_{j=m}^{l}(\ )$ 当 $l<m$ 时其值为 0. 譬如 $\sum_{j=1}^{0}(\ )$ 和 $\sum_{j=n}^{n-1}(\ )$ 都是虚设的项,可删除.

由此看出，求解一般方程组的矩阵分解方法，其设计机理与设计方法同三对角方程组的追赶法（参看本章 6.2.2 小节）如出一辙。

6.3.3 矩阵的 Doolittle 分解

对比 $A=LU_1$ 还可以再考虑 $A=L_1U$ 的分解方式，这里 U 为上三角阵，L_1 为单位下三角阵，如对三阶矩阵 A 有

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & & 0 \\ l_{21} & 1 & \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ & u_{22} & u_{23} \\ 0 & & u_{33} \end{bmatrix}$$

仍按矩阵乘法规则展开，有

$$\begin{aligned} &a_{11}=\underset{\sim}{u_{11}}, & &a_{12}=\underset{\sim}{u_{12}}, & &a_{13}=\underset{\sim}{u_{13}} \\ &a_{21}=\underset{\sim}{l_{21}}u_{11}, & &a_{22}=l_{21}u_{12}+\underset{\sim}{u_{22}}, & &a_{23}=l_{21}u_{13}+\underset{\sim}{u_{23}} \\ &a_{31}=\underset{\sim}{l_{31}}u_{11}, & &a_{32}=l_{31}u_{12}+\underset{\sim}{l_{32}}u_{22}, & &a_{33}=l_{31}u_{13}+l_{32}u_{23}+\underset{\sim}{u_{33}} \end{aligned}$$

容易看出，这里仍可逐行生成 L_1 与 U 的各个元素（波纹线所示）

$$\begin{aligned} &u_{11},u_{12},u_{13} \\ &\to l_{21}\to u_{22},u_{23} \\ &\to l_{31},l_{32}\to u_{33} \end{aligned}$$

其计算公式为

$$\begin{aligned} &u_{11}=a_{11}, & &u_{12}=a_{12}, & &u_{13}=a_{13} \\ &l_{21}=a_{21}/u_{11}, & &u_{22}=a_{22}-l_{21}u_{12}, & &u_{23}=a_{23}-l_{21}u_{13} \\ &l_{31}=a_{31}/u_{11}, & &l_{32}=(a_{32}-l_{31}u_{12})/u_{22}, & &u_{33}=a_{33}-l_{31}u_{13}-l_{32}u_{23} \end{aligned}$$

这一事实同样有普遍意义。对于一般的 n 阶矩阵 A，设将它分解为 $A=L_1U$ 的形式，这里 U 为上三角阵而 L_1 为单位下三角阵，则所给方程组 $Ax=b$ 即 $L_1(Ux)=b$ 可化归为单位下三角方程组 $L_1y=b$ 与上三角方程组 $Ux=y$ 来求解。分解方式 $A=L_1U$ 称作**矩阵 A 的 Doolittle 分解**。

如有兴趣，请读者自行导出 Doolittle 分解的计算公式。

6.4 Cholesky 方法

6.4.1 对称阵的 LL^T 分解

称矩阵 A 是**对称的**，如果其转置阵 $A^T=A$。系数阵为对称阵的线性方程组称作**对称方程组**。

由于三角方程组的求解是简单的,自然希望将对称方程组 $\boldsymbol{Ax}=\boldsymbol{b}$ 加工成三角方程组来求解,为此需要将系数矩阵 $\boldsymbol{A}$ 分解成下三角阵 $\boldsymbol{L}$ 与上三角阵 $\boldsymbol{U}$ 的乘积 $\boldsymbol{A}=\boldsymbol{LU}$(参看图 6.5). 这时由 $\boldsymbol{A}^{\mathrm{T}}=\boldsymbol{A}$ 有 $\boldsymbol{U}^{\mathrm{T}}\boldsymbol{L}^{\mathrm{T}}=\boldsymbol{LU}$,因此应取 $\boldsymbol{U}=\boldsymbol{L}^{\mathrm{T}}$,即令

$$\boldsymbol{A}=\boldsymbol{LL}^{\mathrm{T}}$$

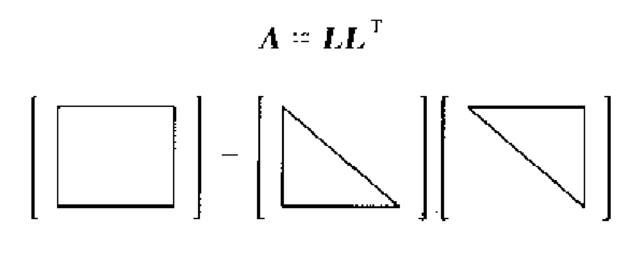


图 6.5 $\boldsymbol{LU}$ 分解示意图

这种设计方法是否合适呢? 对于 $n=1$ 的平凡情形,上述分解公式退化为

$$a=l\cdot l$$

的形式,据此知 $l=\sqrt{a}$. 由此可见矩阵分解 $\boldsymbol{A}=\boldsymbol{LL}^{\mathrm{T}}$中含有开方运算. 对称阵 $\boldsymbol{A}$ 的 $\boldsymbol{LL}^{\mathrm{T}}$分解因此被称为**平方根法**.

平方根法由于含有开方运算而没有实用价值,其矩阵分解过程留做习题供读者自行练习.

6.4.2 对称阵的 Cholesky 分解

为避免开方运算,可采取如下分解方案

$$\boldsymbol{A}=\boldsymbol{L}_1\boldsymbol{DL}_1^{\mathrm{T}}$$

这里 $\boldsymbol{D}$ 为对角阵,$\boldsymbol{L}_1$ 为单位下三角阵,矩阵分解 $\boldsymbol{A}=\boldsymbol{L}_1\boldsymbol{DL}_1^{\mathrm{T}}$如图 6.6 所示,图中齿形线表示对角元素全为 1. 对称阵的这种分解方式称作 **Cholesky 分解**.

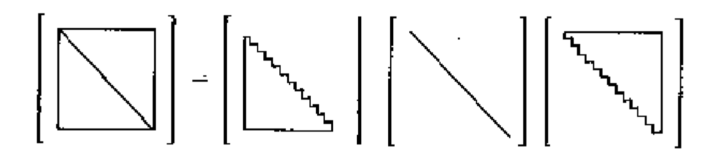


图 6.6 Cholesky 分解示意图

这种设计方法是否有效呢? 对于 $n=1$ 的平凡情形,分解公式 $\boldsymbol{A}=\boldsymbol{L}_1\boldsymbol{DL}_1^{\mathrm{T}}$ 退化为

$$a=1\cdot d\cdot 1$$

的形式,据此立即定出 $d=a$,这里确实不含开方运算.

为具体显示这种分解过程,再考察三阶矩阵的情形:

$$\begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{21} & a_{22} & a_{32} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & & \mathbf{0} \\ l_{21} & 1 & \\ l_{31} & l_{32} & 1 \end{bmatrix} \times \begin{bmatrix} d_1 & & \mathbf{0} \\ & d_2 & \\ \mathbf{0} & & d_3 \end{bmatrix} \times \begin{bmatrix} 1 & l_{21} & l_{31} \\ & 1 & l_{32} \\ \mathbf{0} & & 1 \end{bmatrix}$$

按矩阵乘法规则展开,注意到所给矩阵的对称性可列出方程组

$$a_{11} = d_1$$

$$a_{21} = d_1 l_{21}, \quad a_{22} = d_1 l_{21}^2 + d_2$$

$$a_{31} = d_1 l_{31}, \quad a_{32} = d_1 l_{21} l_{31} + d_2 l_{32}, \quad a_{33} = d_1 l_{31}^2 + d_2 l_{32}^2 + d_3$$

据此可逐行求出分解阵 $\boldsymbol{L}_1$ 与 $\boldsymbol{D}$ 的各个元素:

$$d_1 = a_{11}$$

$$l_{21} = a_{21}/d_1, \quad d_2 = a_{22} - d_1 l_{21}^2$$

$$l_{31} = a_{31}/d_1, \quad l_{32} = (a_{32} - d_1 l_{21} l_{31})/d_2, \quad d_3 = a_{33} - d_1 l_{31}^2 - d_2 l_{32}^2$$

进一步推广到 n 阶方阵的一般情形. 这时对称阵的 Cholesky 分解 $\boldsymbol{A} = \boldsymbol{L}_1 \boldsymbol{D} \boldsymbol{L}_1^{\mathrm{T}}$ 具有形式

$$\begin{bmatrix} a_{11} & & \text{对} & & \\ a_{21} & a_{22} & & \text{称} & \\ a_{31} & a_{32} & a_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} =$$

$$\begin{bmatrix} 1 & & & & \mathbf{0} \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ \vdots & \vdots & \ddots & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{n,n-1} & 1 \end{bmatrix} \times \begin{bmatrix} d_1 & & & \mathbf{0} \\ & d_2 & & \\ & & d_3 & \\ & & & \ddots & \\ \mathbf{0} & & & & d_n \end{bmatrix} \times \begin{bmatrix} 1 & l_{21} & l_{31} & \cdots & l_{n1} \\ & 1 & l_{32} & \cdots & l_{n2} \\ & & \ddots & \ddots & \vdots \\ & & & 1 & l_{n,n-1} \\ \mathbf{0} & & & & 1 \end{bmatrix}$$

将上式按矩阵乘法规则展开,左端的元素 $a_{ij}(j \leqslant i)$ 等于 $\boldsymbol{L}_1$ 的第 i 行与 $\boldsymbol{D}\boldsymbol{L}_1^{\mathrm{T}}$ 的第 j 列的乘积

$$\begin{aligned} a_{ij} &= (l_{i1}, l_{i2}, \cdots, l_{i,j-1}, l_{ij}, l_{i,j+1}, \cdots, l_{i,i-1}, 1, 0, \cdots, 0) \times \\ &\quad (d_1 l_{j1}, d_2 l_{j2}, \cdots, d_{j-1} l_{j,j-1}, d_j, 0, \cdots, 0)^{\mathrm{T}} \\ &= \sum_{k=1}^{j-1} d_k l_{ik} l_{jk} + l_{ij} d_j \end{aligned}$$

$$a_{ii} = \sum_{k=1}^{i-1} d_k l_{ik}^2 + d_i$$

据此可逐行定出分解阵 $\boldsymbol{L}_1$ 与 $\boldsymbol{D}$ 的元素

$$\begin{cases} l_{ij} = \left(a_{ij} - \sum_{k=1}^{j-1} d_k l_{ik} l_{jk}\right)/d_j, & j = 1,2,\cdots,i-1 \\ d_i = a_{ii} - \sum_{k=1}^{i-1} d_k l_{ik}^2, & i = 1,2,\cdots,n \end{cases} \tag{13}$$

由此可见，同对称阵的 LL^T 分解不同，上述 Cholesky 分解确实不再含有开方运算.

可以证明，如果所给对称阵 A 是所谓正定阵①，那么分解公式(13)的除数 d_i 全不为 0，这时 Cholesky 分解的计算过程不会中断.

基于对称正定阵 A 的 Cholesky 分解 $A = L_1 D L_1^T$，所给方程组 $Ax = b$ 即

$$L_1(DL_1^T x) = b$$

化归为如下两个三角方程组

$$L_1 y = b;\ L_1^T x = D^{-1} y$$

其求解公式分别为

$$y_i = b_i - \sum_{j=1}^{i-1} l_{ij} y_j, \quad i = 1,2,\cdots,n \tag{14}$$

和

$$x_i = y_i/d_i - \sum_{j=i+1}^{n} l_{ji} x_j, \quad i = n, n-1, \cdots, 1 \tag{15}$$

基于 Cholesky 分解 $A = L_1 D L_1^T$ 求解对称方程组 $Ax = b$ 的这种方法通常称作 **Cholesky 方法**.

算法 6.3（对称方程组的 Cholesky 方法）

1. 预处理 施行矩阵分解 $A = L_1 D L_1^T$，即依式(13)对 $i = 1,2,\cdots,n$ 依次计算 $l_{i1}, l_{i2}, \cdots, l_{i,i-1}$ 与 d_i.

2. 追的过程 求解单位下三角方程组 $L_1 y = b$，即依式(14)顺序计算 $y_1, y_2, \cdots, y_n$.

3. 赶的过程 求解单位上三角方程组 $L_1^T x = D^{-1} y$，即依式(15)逆序求解 $x_n, x_{n-1}, \cdots, x_1$.

不难知道，运用 Cholesky 方法求解 n 阶对称正定方程组，其总运算量约为 $\frac{1}{6}n^3$ 次乘除操作.

① 正定阵的定义可参看高等代数或矩阵论的有关书籍.

6.4.3　对称阵的压缩存储技巧

考虑到系数矩阵 **A** 的对称性,可将其下三角部分

$$\begin{matrix} a_{11} \\ a_{21}\ a_{22} \\ \cdots\ \cdots\ \cdots\ \cdots \\ a_{n1}\ a_{n2}\ \cdots\ a_{nn} \end{matrix}$$

逐行存放于一维数组 $\bar{A}[1:m]$ 中. 现确定数组 $\bar{A}$ 与矩阵 **A** 两者元素的对应关系.

首先注意一个明显的事实:数组 $\bar{A}$ 中元素的个数为

$$m = 1+2+\cdots+n = \frac{n(n+1)}{2}$$

设用 k 表示 **A** 的下三角部分前 $i-1$ 行元素的总数,则

$$k = \frac{i(i-1)}{2}$$

于是,矩阵元素 a_{ij} 在数组 $\bar{A}$ 中显然是第 $k+j$ 个元素,即 $a_{ij}\ (i \ge j)$ 存放于 $\bar{A}[k+j]$ 中. 考虑到这一事实,不难写出压缩存储的 Cholesky 方法的计算程序.

6.5　消去法

正如 6.1 节和 6.2 节处理三对角方程组所看到的那样,线性方程组的求解既可以运用矩阵分解方法,亦可直接借助于人们所熟悉的消元手续. 这两种方法其实是等价的. 6.3 节已讨论过一般方程组的矩阵分解方法,下面再针对一般方程组考察消元手续.

6.5.1　Gauss 消去法的设计思想

人们都很熟悉求解线性方程组的消去法. 消去法是一种古老的方法,但用在现代计算机上依然十分有效.

消去法的设计思想是,通过将一个方程乘或除以某个常数,以及将两个方程相加减这两种手续,逐步消去方程中的变元,而将所给方程组加工成便于求解的三角方程组乃至对角方程组的形式.

首先考察三阶方程组的情形:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \end{cases} \tag{16}$$

现在逐行施行消元手续. 第 1 步,先将(16)$_1$ 中 x_1 的系数化为 1,使之变成

$$x_1 + a_{12}^{(1)} x_2 + a_{13}^{(1)} x_3 = b_1^{(1)}$$

然后利用它从(16)的其余方程中消去 x_1,归结为关于变元 x_2,x_3 的二阶方程组

$$\begin{cases} a_{22}^{(1)} x_2 + a_{23}^{(1)} x_3 = b_2^{(1)} \\ a_{32}^{(1)} x_2 + a_{33}^{(1)} x_3 = b_3^{(1)} \end{cases} \tag{17}$$

第 2 步,再将(17)$_1$ 中 x_2 的系数化为 1,使之变成

$$x_2 + a_{23}^{(2)} x_3 = b_2^{(2)}$$

然后利用它从方程(17)$_2$ 中消去 x_2,结果求出

$$x_3 = a_3^{(3)}$$

这样,所给方程组(16)被加工成如下形式:

$$\begin{cases} x_1 + a_{12}^{(1)} x_2 + a_{13}^{(1)} x_3 = b_1^{(1)} \\ \qquad\quad x_2 + a_{23}^{(2)} x_3 = b_2^{(2)} \\ \qquad\qquad\qquad x_3 = b_3^{(3)} \end{cases}$$

这是一个单位上三角方程组,如前所述,通过回代过程容易求出它的解.

解线性方程组上述方法,其基本思想是将所给线性方程组通过消元手续加工成单位上三角方程组. 这种方法称作 **Gauss 消去法**.

6.5.2 Gauss 消去法的计算步骤

进而考察一般形式的线性方程组

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1, 2, \cdots, n \tag{18}$$

其 Gauss 消去法分消元过程与回代过程两个环节:

(一) 消元过程

第 1 步,将方程(18)$_1$ 中变元 x_1 的系数化为 1,使之变成

$$x_1 + \sum_{j=2}^{n} a_{1j}^{(1)} x_j = b_1^{(1)}$$

式中

$$a_{1j}^{(1)} = a_{1j}/a_{11}, \quad j = 2, 3, \cdots, n$$
$$b_1^{(1)} = b_1/a_{11}$$

然后利用它从(18)的其余方程中消去 x_1,将它们加工成关于变元 $x_2, x_3, \cdots, x_n$ 的 $n-1$ 阶方程组(较原方程组降了一阶)

$$\sum_{j=2}^{n} a_{ij}^{(1)} x_j = b_i^{(1)}, \quad i = 2, 3, \cdots, n \tag{19}$$

为此所要施行的运算手续是

$$a_{ij}^{(1)} = a_{ij} - a_{i1}a_{1j}^{(1)},\quad b_i^{(1)} = b_i - a_{i1}b_1^{(1)},\qquad i,j=2,3,\cdots,n$$

如此继续下去，这样经过 $k-1$ 步消元以后，依次得出 $k-1$ 个方程

$$x_i + \sum_{j=i+1}^{n} a_{ij}^{(i)}x_j = b_i^{(i)},\quad i=1,2,\cdots,k-1$$

和关于变元 $x_j(j=k,k+1,\cdots,n)$ 的 $n-k+1$ 阶方程组

$$\sum_{j=k}^{n} a_{ij}^{(k-1)}x_j = b_i^{(k-1)},\quad i=k,k+1,\cdots,n \tag{20}$$

第 k 步进一步将方程组(20)加工成

$$\begin{cases} x_k + \sum\limits_{j=k+1}^{n} a_{kj}^{(k)}x_j = b_k^{(k)} \\ \sum\limits_{j=k+1}^{n} a_{ij}^{(k)}x_j = b_i^{(k)},\quad i=k+1,k+2,\cdots,n \end{cases}$$

的形式，式中

$$\begin{aligned} a_{kj}^{(k)} &= a_{kj}^{(k-1)}/a_{kk}^{(k-1)},\quad j=k+1,k+2,\cdots,n \\ b_k^{(k)} &= b_k^{(k-1)}/a_{kk}^{(k-1)} \end{aligned} \tag{21}$$

而

$$\begin{aligned} a_{ij}^{(k)} &= a_{ij}^{(k-1)} - a_{ik}^{(k-1)}a_{kj}^{(k)}, \\ b_i^{(k)} &= b_i^{(k-1)} - a_{ik}^{(k-1)}b_k^{(k)}, \end{aligned}\qquad i,j=k+1,k+2,\cdots,n \tag{22}$$

上述消元手续做 n 步以后，所给方程组(18)被加工成如下形式

$$x_i + \sum_{j=i+1}^{n} a_{ij}^{(i)}x_j = b_i^{(i)},\quad i=1,2,\cdots,n \tag{23}$$

(二) 回代过程

方程组(23)即

$$\begin{cases} x_1 + a_{12}^{(1)}x_2 + a_{13}^{(1)}x_3 + \cdots + a_{1n}^{(1)}x_n = b_1^{(1)} \\ \qquad x_2 + a_{23}^{(2)}x_3 + \cdots + a_{2n}^{(2)}x_n = b_2^{(2)} \\ \qquad\qquad \cdots\ \cdots\ \cdots\ \cdots \\ \qquad\qquad x_{n-1} + a_{n-1,n}^{(n-1)}x_n = b_{n-1}^{(n-1)} \\ \qquad\qquad\qquad x_n = b_n^{(n)} \end{cases}$$

的求解很方便，自下而上逐步回代即得所求的解

$$\begin{cases} x_n = b_n^{(n)} \\ x_i = b_i^{(i)} - \sum\limits_{j=i+1}^{n} a_{ij}^{(i)}x_j,\quad i=n-1,n-2,\cdots,1 \end{cases} \tag{24}$$

算法 6.4(线性方程组的 Gauss 消去法)

步 1 对 $k=1,2,\cdots,n$ 反复执行算式(21),(22),定出方程组(23)的系数 $a_{ij}^{(i)}$, $b_i^{(i)}$.

步 2 依据式(24)求出解 x_i.

图 6.7 描述了 Gauss 消去法,其中框 1 的含义后文 6.5.3 小节将给出说明.

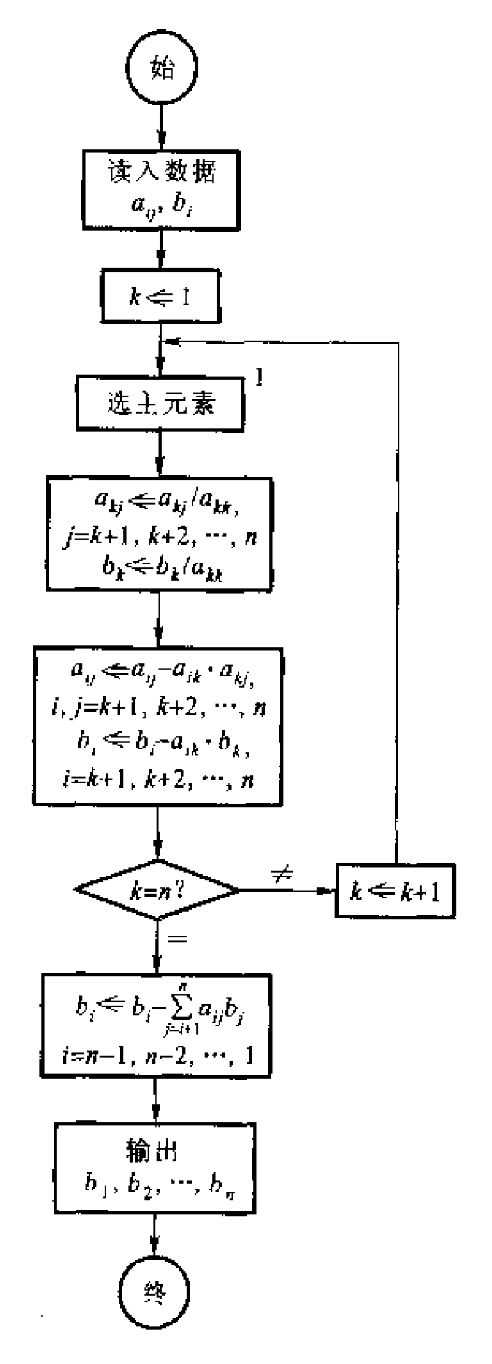


图 6.7 Gauss 消去法的计算流程

现在统计 Gauss 消去法的计算量.如引论 0.2 节所指出的,计算机上乘除操

作通常比加减操作耗时多，因此，如果加减运算的次数同乘除运算的次数相差不大，可以只统计乘除运算的次数算作计算量．对于 Gauss 消去法，其消元过程的计算量为

$$\sum_{k=1}^{n-1}[(n-k)^2+2(n-k)]=\frac{1}{3}n^3+\frac{1}{2}n^2-\frac{5}{6}n$$

而回代过程的计算量为

$$\sum_{k=1}^{n}(n-k+1)=\frac{1}{2}n^2+\frac{1}{2}n$$

由此可见，当 n 充分大时，Gauss 消去法的总计算量约为 $\frac{1}{3}n^3$ 次乘除运算．

值得指出的是，线性方程组的两种解法——矩阵分解法与 Gauss 消去法其实是殊途同归．不难看出，这里归结出的式(23)其实就是 6.3 节 Crout 分解中的上三角方程组 $\boldsymbol{U}_1\boldsymbol{x}=\boldsymbol{y}$．

6.5.3　选主元素

再考察 Gauss 消去法的消元过程，可以看到，其第 k 步要用 $a_{kk}^{(k-1)}$ 做除法，这就要求保证它们全不为 0．什么样的矩阵能保证满足这项要求呢？

上一章 5.3.3 小节已介绍过对角占优阵的概念．称 n 阶方阵 $\boldsymbol{A}=(a_{ij})_{n\times n}$ 是**对角占优**的，如果其主对角元素的绝对值大于同行其他元素绝对值之和：

$$|a_{ii}|>\sum_{\substack{j=1\\j\neq i}}^{n}|a_{ij}|,\quad i=1,2,\cdots,n$$

定理 2　如果方程组(18)是对角占优的，则按式(21)，式(22)求出的 $a_{kk}^{(k-1)}(k=1,2,\cdots,n)$ 全不为 0．

证　先考察消元过程的第 1 步．因方程组(18)为对角占优，有

$$|a_{11}|>\sum_{j=2}^{n}|a_{1j}| \tag{25}$$

故 $a_{11}^{(0)}=a_{11}\neq 0$．又据式(21)，式(22)知

$$a_{ij}^{(1)}=a_{ij}-\frac{a_{i1}a_{1j}}{a_{11}},\quad i,j=2,3,\cdots,n \tag{26}$$

于是

$$\sum_{\substack{j=2\\j\neq i}}^{n}|a_{ij}^{(1)}|\leqslant\sum_{\substack{j=2\\j\neq i}}^{n}|a_{ij}|+\frac{|a_{i1}|}{|a_{11}|}\sum_{\substack{j=2\\j\neq i}}^{n}|a_{1j}|$$

$$=\sum_{\substack{j=1\\j\neq i}}^{n}|a_{ij}|-|a_{i1}|+\frac{|a_{i1}|}{|a_{11}|}\left(\sum_{j=2}^{n}|a_{1j}|-|a_{1i}|\right)$$

再利用所给方程组的对角占优性，由上式可进一步得

$$\sum_{\substack{j=2\\ j\neq i}}^{n} |a_{ij}^{(1)}| < |a_{ii}| - |a_{i1}| + \frac{|a_{i1}|}{|a_{11}|}(|a_{11}| - |a_{1i}|)$$

$$= |a_{ii}| - \frac{|a_{i1}||a_{1i}|}{|a_{11}|}$$

又据式(26)

$$|a_{ii}^{(1)}| = \left| a_{ii} - \frac{a_{i1}a_{1i}}{a_{11}} \right| \geqslant |a_{ii}| - \frac{|a_{i1}||a_{1i}|}{|a_{11}|}$$

故有

$$\sum_{\substack{j=2\\ j\neq i}}^{n} |a_{ij}^{(1)}| < |a_{ii}^{(1)}|, \quad i = 2,3,\cdots,n$$

这说明消元过程第 1 步所归结出的方程组(19)同样是对角占优的. 从而又有 $a_{22}^{(1)} \neq 0$. 依此类推即可断定一切 $a_{kk}^{(k-1)}$ 全不为 0. 证毕.

一般线性方程组使用 Gauss 消去法求解时，即使 $a_{kk}^{(k-1)}$ 不为 0，但如果其绝对值很小，舍入误差的影响也会严重地损失精度. 实际计算时必须预防这类情况发生.

例 考察方程组

$$\begin{cases} 10^{-5}x_1 + x_2 = 1 \\ x_1 + x_2 = 2 \end{cases} \tag{27}$$

设用 Gauss 消去法求解. 先用 10^{-5} 除 $(27)_1$，然后利用它从 $(27)_2$ 中消去 x_1，得

$$\begin{cases} x_1 + 10^5 x_2 = 10^5 \\ (1 - 10^5)x_2 = 2 - 10^5 \end{cases} \tag{28}$$

设取 4 位浮点十进制进行计算，以≈表示**对阶舍入**的计算过程，则有

$$1 - 10^5 \approx -10^5,\ 2 - 10^5 \approx -10^5$$

因而这时方程组(28)的实际形式是

$$\begin{cases} x_1 + 10^5 x_2 = 10^5 \\ x_2 = 1 \end{cases}$$

由此回代解出 $x_1 = 0, x_2 = 1$.

这个结果严重失真，究其根源，是由于所用的除数太小，使得方程 $(28)_1$ 在消元过程中“吃掉”了方程 $(27)_2$. 避免这类错误的一种有效方法是，在消元前先调整方程的次序. 设将方程组(27)改写为

$$\begin{cases} x_1 + x_2 = 2 \\ 10^{-5}x_1 + x_2 = 1 \end{cases}$$

再进行消元，得

$$\begin{cases} x_1 + x_2 = 2 \\ (1-10^{-5})x_2 = 1 - 2\times 10^{-5} \end{cases}$$

这里 $1-10^{-5}\approx 1, 1-2\times 10^{-5}\approx 1$，因而上述方程组的实际形式是

$$\begin{cases} x_1 + x_2 = 2 \\ x_2 = 1 \end{cases}$$

由此回代解出 $x_1 = x_2 = 1$. 这个结果是正确的.

可以在 Gauss 消去法的消元过程中运用上述技巧. 为此再考察第 k 步所要加工的方程组(20). 检查其中变元 x_k 的各个系数 $a_{kk}^{(k-1)}, a_{k+1,k}^{(k-1)}, \cdots, a_{nk}^{(k-1)}$，从中

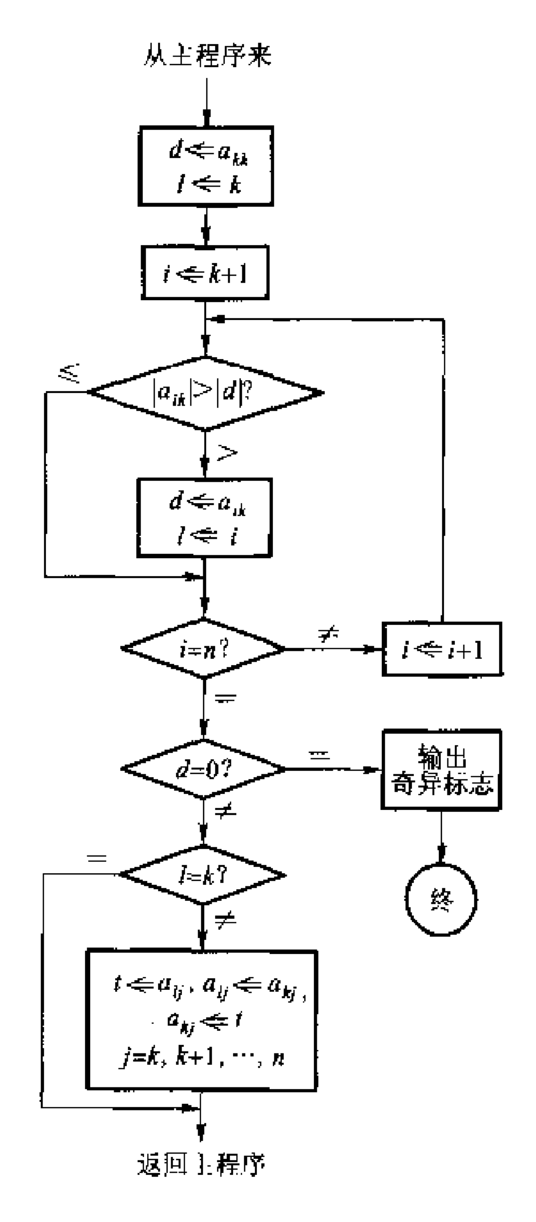


图 6.8　选主元素的处理过程

挑选出绝对值最大的一个，称之为第 k 步的**主元素**[1]. 设主元素在第 $l(k\le l\le n)$ 个方程，即 $|a_{lk}^{(k-1)}| = \max\limits_{k\le i\le n}|a_{ik}^{(k-1)}|$，若 $l\neq k$，则先将第 l 个方程与第 k 个方程互易位置，使得新的 $a_{kk}^{(k-1)}$ 成为主元素，然后再着手消元，这一手续称作**选主元素**.

图 6.8 描述了选主元素的处理过程，它是图 6.7 中框 1 的具体化.

最后讨论前述 Gauss 消去法的可行性.

定理 3 设所给方程组(18)对称并且是对角占优的，则 $a_{kk}^{(k-1)}(k=1,2,\cdots,n)$全是主元素.

证 因方程组(18)对称且为对角占优，据式(24)有

$$|a_{11}| > \sum_{i=2}^{n}|a_{i1}| \geqslant \max_{2\le i\le n}|a_{i1}|$$

故 a_{11} 是主元素. 再由式(26)有

$$a_{ij}^{(1)} = a_{ij} - \frac{a_{i1}a_{1j}}{a_{11}}$$

$$= a_{ji} - \frac{a_{1i}a_{j1}}{a_{11}} = a_{ji}^{(1)}, \quad i,j=2,3,\cdots,n$$

因而所归结出的方程组(19)也是对称的. 不难证明它也是对角占优的. 故 $a_{22}^{(1)}$ 也是主元素. 依此类推，知一切 $a_{kk}^{(k-1)}$ 全是主元素.

6.6 中国古代数学的"方程术"

《九章算术》是我国数学史上一部重要的数学经典，据考证它成书于秦汉时期，距今已有两千年了. 该书共分九章，其中第八章名为"方程"章. 下面列出的"禾实问题"是方程章中一道数学题:

今有上禾三秉，中禾二秉，下禾一秉，实三十九斗. 上禾二秉，中禾三秉，下禾一秉，实三十四斗. 上禾一秉，中禾二秉，下禾三秉，实二十六斗. 问:上、中、下三禾一秉各几何?

答曰:上禾一秉九斗四分斗之一. 中禾一秉四斗四分斗之一. 下禾一秉二斗四分斗之三.

翻译成白话文，"禾实问题"是说:

现有上等稻 3 捆，中等稻 2 捆，下等稻 1 捆，共得谷 39 斗；上等稻 2 捆，中等稻 3 捆，下等稻 1 捆，共得谷 34 斗；上等稻 1 捆，中等稻 2 捆，下等稻 3 捆，共得

1 这样选出的主元素通常称作**列主元素**. 在编制实用程序时，常在方程组(20)的所有系数 $a_{ij}^{(k-1)}$ $(i,j=k,k+1,\cdots,n)$中选取一个绝对值最大者，称之为第 k 步的**全主元素**.

谷 26 斗. 问上等、中等、下等三种稻每捆各得谷多少?

答:上等稻每捆得谷 $9\frac{1}{4}$ 斗,中等稻每捆 $4\frac{1}{4}$ 斗,下等稻每捆 $2\frac{3}{4}$ 斗.

翻译成现代数学的语言,禾实问题告诉人们,线性(联立)方程组

$$\begin{cases}3x+2y+z=39\\2x+3y+z=34\\x+2y+3z=26\end{cases}\tag{29}$$

的解是 $x=9\frac{1}{4}, y=4\frac{1}{4}, z=2\frac{3}{4}$.

早在两千多年前,智慧的中国先民是怎样得出这个结果的呢?

中国古代数学的一个重要特色是注重实际计算. 算经《九章算术》充分体现了这一特色.《九章算术》常常采取这样的写法:先提出问题,然后给出答案,最后归纳总结出所谓"术".《九章算术》中的"术"实际上是程序化的解法,也就是"算法". 在某种意义上,中国古代数学中的"算术"就是算法设计技术.

在古代中国,计算过程是通过布置和摆弄算筹来完成的. 按照《九章算术》所给出的"方程术",上述禾实问题的筹算过程大意如下:

$$\begin{array}{cccccc} & \text{上等} & \text{中等} & \text{下等} & & \text{稻谷} \end{array}$$

$$\begin{bmatrix}3&2&1&\vdots&39\\2&3&1&\vdots&34\\1&2&3&\vdots&26\end{bmatrix}\begin{matrix}①\\②\\③\end{matrix}$$

$$\xrightarrow{②\times3\quad ①\times2}\begin{bmatrix}3&2&1&\vdots&39\\0&5&1&\vdots&24\\1&2&3&\vdots&26\end{bmatrix}\begin{matrix}①\\②\\③\end{matrix}$$

$$\xrightarrow{③\times3-①}\begin{bmatrix}3&2&1&\vdots&39\\0&5&1&\vdots&24\\0&4&8&\vdots&39\end{bmatrix}\begin{matrix}①\\②\\③\end{matrix}$$

$$\xrightarrow{③\times5-②\times4}\begin{bmatrix}3&2&1&\vdots&39\\0&5&1&\vdots&24\\0&0&36&\vdots&99\end{bmatrix}$$

这样,通过上述加工手续,所给方程组(29)被加工成如下形式的上三角方程组

$$\begin{cases}3x+2y+z=39\\\quad\ 5y+z=24\\\quad\quad 36z=99\end{cases}$$

如前所述,这种特殊形式的方程组通过回代过程容易求出它的解.

值得强调指出的是,《九章算术》这部中国古代算经,其中最为引人注目的数

学成就之一是，它在世界数学史上最早提出了求解线性方程组的概念，并且系统地总结出线性方程组的程序化解法——所谓“方程术”.

令人不可思议的是，中国古代先哲早在两千多年前所提出的“方程术”，其设计方法竟类同于近代求解线性方程组的 Gauss 消去法.

小　　结

作为本章核心内容的求解三对角方程组的追赶法，将三对角方程组的求解过程，加工成下二对角方程组与上二对角方程组两个简单求解过程的重叠，其中，下二对角方程组的求解是个顺序前进的追的过程，而上二对角方程组的求解则是个逆序后退的赶的过程. 一下一上，一顺一逆，一追一赶，一进一退，相反相成.

追赶法的设计思想对一般形式的线性方程组 $\boldsymbol{Ax}=\boldsymbol{b}$ 同样是有效的. 设记 $\boldsymbol{L},\boldsymbol{U}$ 为下三角阵与上三角阵，$\boldsymbol{L}_1$ 与 $\boldsymbol{U}_1$ 分别表示单位下三角阵与单位上三角阵，那么矩阵 $\boldsymbol{A}$ 的三角分解 $\boldsymbol{A}=\boldsymbol{LU}$ 有 Crout 分解与 Doolittle 分解两种方式，它们分别将前后两个三角阵 $\boldsymbol{L}$ 与 $\boldsymbol{U}$ 单位化，即其分解方式分别为 $\boldsymbol{A}=\boldsymbol{LU}_1$ 与 $\boldsymbol{A}=\boldsymbol{L}_1\boldsymbol{U}$. 这两种分解方式是互反的.

矩阵三角分解的两种方式 $\boldsymbol{A}=\boldsymbol{LU}_1$ 与 $\boldsymbol{A}=\boldsymbol{L}_1\boldsymbol{U}$ 又可统一地表达为 $\boldsymbol{A}=\boldsymbol{L}_1\boldsymbol{DU}_1$ 的形式，这里 $\boldsymbol{D}$ 为对角阵，而 $\boldsymbol{L}_1,\boldsymbol{U}_1$ 分别为单位下三角阵与单位上三角阵. 这样，矩阵 $\boldsymbol{A}$ 的 Crout 分解与 Doolittle 分解可分别表现为 $\boldsymbol{A}=(\boldsymbol{L}_1\boldsymbol{D})\boldsymbol{U}_1$ 与 $\boldsymbol{A}=\boldsymbol{L}_1(\boldsymbol{DU}_1)$.

比较矩阵分解技术 $\boldsymbol{A}=\boldsymbol{L}_1\boldsymbol{DU}_1$ 与上一章的矩阵分裂技术 $\boldsymbol{A}=\boldsymbol{L}_0+\boldsymbol{D}+\boldsymbol{U}_0$ 是有趣的：前者用矩阵乘法，后者用矩阵加法，前者 $\boldsymbol{L}_1,\boldsymbol{U}_1$ 的主对角元素全为 1，后者 $\boldsymbol{L}_0,\boldsymbol{U}_0$ 的主对角元素全为 0. 可见这两种处理手续互为反手续.

在这种意义上可以认为，求解线性方程组的直接法和迭代法互为反方法.

题解 6.1　三对角方程组的“赶追法”

提要　追赶法的设计思想是将所给三对角方程组加工成单位上二对角方程组来求解. 自然会问，三对角方程组能否通过消元手续或矩阵分解手续加工成单位下二对角方程组？就是说，对应于追赶法，能否设计出从逆序（赶的过程）到顺序（追的过程）的“赶追法”？

回答是肯定的. 现在就三阶三对角方程组

$$\begin{cases} b_1x_1+c_1x_2=f_1 \\ a_2x_1+b_2x_2+c_2x_3=f_2 \\ a_3x_2+b_3x_3=f_3 \end{cases}$$

揭示赶追法的概貌，赶追法是要将它加工成如下形式的单位下二对角方程组

$$\begin{cases} x_1=y_1 \\ l_2x_1+x_2=y_2 \\ l_3x_2+x_3=y_3 \end{cases}$$

后者是容易求解的.

加工手续类同于6.1节的追赶法.这里计算流程分下列三个环节：

1. **预处理** 交替生成 $d_3\to l_3\to d_2\to l_2\to d_1$：

$$d_3=b_3,\ l_3=a_3/d_3$$
$$d_2=b_2-c_2l_3,\ l_2=a_2/d_2$$
$$d_1=b_1-c_1l_2$$

2. **赶的过程** 逆序计算 $y_3\to y_2\to y_1$：

$$y_3=f_3/d_3$$
$$y_2=(f_2-c_2y_3)/d_2$$
$$y_1=(f_1-c_1y_2)/d_1$$

3. **追的过程** 顺序求解 $x_1\to x_2\to x_3$：

$$x_1=y_1$$
$$x_2=y_2-l_2x_1$$
$$x_3=y_3-l_3x_2$$

上述处理过程不难推广到 n 阶三对角方程组的一般情形.

再从矩阵分解的角度进行考察.设所给三对角阵 $\boldsymbol{A}$ 可进行二对角分解 $\boldsymbol{A}=\boldsymbol{U}\boldsymbol{L}_1$，这里 $\boldsymbol{U}$ 为上二对角阵，$\boldsymbol{L}_1$ 为单位下二对角阵，则所给方程组 $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{f}$ 即 $\boldsymbol{U}(\boldsymbol{L}_1\boldsymbol{x})=\boldsymbol{f}$ 可化归为两个二对角方程组 $\boldsymbol{U}\boldsymbol{y}=\boldsymbol{f}$，$\boldsymbol{L}_1\boldsymbol{x}=\boldsymbol{y}$ 来求解，它们的求解过程分别是逆序的赶的过程与顺序的追的过程，因此这种方法可称为**赶追法**.

求解三对角方程时，如果追赶法的计算过程中断则可尝试改用赶追法.

题1 给定三对角阵

$$\boldsymbol{A}=\begin{bmatrix} b_1 & c_1 & & & \boldsymbol{0} \\ a_2 & b_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-1} & b_{n-1} & c_{n-1} \\ \boldsymbol{0} & & & a_n & b_n \end{bmatrix}$$

试将 $\boldsymbol{A}$ 分解为上二对角阵 $\boldsymbol{U}$ 与单位下二对角阵 $\boldsymbol{L}_1$ 的乘积 $\boldsymbol{A}=\boldsymbol{U}\boldsymbol{L}_1$.

解 按矩阵乘法规则展开 $\boldsymbol{A}=\boldsymbol{U}\boldsymbol{L}_1$:

$$\begin{bmatrix} b_1 & c_1 & & & \boldsymbol{0} \\ a_2 & b_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & a_{n-1} & b_{n-1} & c_{n-1} \\ \boldsymbol{0} & & & a_n & b_n \end{bmatrix} = \begin{bmatrix} d_1 & u_1 & & & \boldsymbol{0} \\ & d_2 & u_2 & & \\ & & \ddots & \ddots & \\ & & & d_{n-1} & u_{n-1} \\ \boldsymbol{0} & & & & d_n \end{bmatrix} \begin{bmatrix} 1 & & & & \boldsymbol{0} \\ l_2 & 1 & & & \\ & \ddots & \ddots & & \\ & & l_{n-1} & 1 & \\ \boldsymbol{0} & & & l_n & 1 \end{bmatrix}$$

易知 $u_i=c_i,\ i=1,2,\cdots,n-1$,而 d_i 与 l_i 可交替求出:

$$\begin{aligned} & d_n = b_n \\ & l_i = a_i/d_i, \\ & d_{i-1} = b_{i-1} - c_{i-1}l_i, \end{aligned} \qquad i=n,n-1,\cdots,2$$

题 2 设 $\boldsymbol{U}$ 为上二对角阵,$\boldsymbol{L}_1$ 为单位下二对角阵,试列出求解 $\boldsymbol{U}\boldsymbol{y}=\boldsymbol{f}$ 与 $\boldsymbol{L}_1\boldsymbol{x}=\boldsymbol{y}$ 的回代公式.

解 方程组 $\boldsymbol{U}\boldsymbol{y}=\boldsymbol{f}$

$$\begin{bmatrix} d_1 & u_1 & & & \boldsymbol{0} \\ & d_2 & u_2 & & \\ & & \ddots & \ddots & \\ & & & d_{n-1} & u_{n-1} \\ \boldsymbol{0} & & & & d_n \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{n-1} \\ f_n \end{bmatrix}$$

即

$$\begin{cases} d_i y_i + u_i y_{i+1} = f_i, & i=1,2,\cdots,n-1 \\ d_n y_n = f_n \end{cases}$$

其回代计算是个赶的过程:

$$\begin{cases} y_n = f_n/d_n \\ y_i = (f_i - u_i y_{i+1})/d_i, & i=n-1,n-2,\cdots,1 \end{cases}$$

此外,方程组 $\boldsymbol{L}_1\boldsymbol{x}=\boldsymbol{y}$

$$\begin{bmatrix} 1 & & & & \boldsymbol{0} \\ l_2 & 1 & & & \\ & \ddots & \ddots & & \\ & & l_{n-1} & 1 & \\ \boldsymbol{0} & & & l_n & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}$$

即

$$\begin{cases} x_1 = y_1 \\ l_i x_{i-1} + x_i = y_i, \quad i = 2,3,\cdots,n \end{cases}$$

其回代计算是个追的过程：

$$\begin{cases} x_1 = y_1 \\ x_i = y_i - l_i x_{i-1}, \quad i = 2,3,\cdots,n \end{cases}$$

题 3 证明，当前述三对角阵 **A**(见题 1)为对角占优时①上述赶追法是可行的，即一切 $d_i(i=1,2,\cdots,n)$全不为 0.

证 按对角占优条件

$$|d_n| > |a_n|$$

故 $d_n \neq 0$，又

$$|l_n| = \frac{|a_n|}{|d_n|} < 1$$

故

$$\begin{aligned} |d_{n-1}| &= |b_{n-1} - c_{n-1} l_n| \\ &\geqslant |b_{n-1}| - |c_{n-1}||l_n| \\ &\geqslant |b_{n-1}| - |c_{n-1}| \end{aligned}$$

再利用对角占优条件知

$$|d_{n-1}| > |a_{n-1}|$$

因而又有 $d_{n-1} \neq 0$. 其余类推.

题 4 将题 1 的三对角阵 A 分解为 $\boldsymbol{A} = \boldsymbol{L}_1 \boldsymbol{D} \boldsymbol{U}_1$ 的形式，这里 $\boldsymbol{L}_1$、$\boldsymbol{U}_1$ 分别为单位下二对角阵与单位上二对角阵，$\boldsymbol{D}$ 为对角阵.

解 具体表达 $\boldsymbol{A} = \boldsymbol{L}_1 \boldsymbol{D} \boldsymbol{U}_1$ 有

$$\boldsymbol{A} = \begin{bmatrix} 1 & & & & \mathbf{0} \\ l_2 & 1 & & & \\ & \ddots & \ddots & & \\ & & l_{n-1} & 1 & \\ \mathbf{0} & & & l_n & 1 \end{bmatrix} \begin{bmatrix} d_1 & & & & \mathbf{0} \\ & d_2 & & & \\ & & \ddots & & \\ & & & d_{n-1} & \\ \mathbf{0} & & & & d_n \end{bmatrix} \begin{bmatrix} 1 & u_1 & & & \mathbf{0} \\ & 1 & u_2 & & \\ & & \ddots & \ddots & \\ & & & 1 & u_{n-1} \\ \mathbf{0} & & & & 1 \end{bmatrix}$$

按矩阵乘法规则展开，注意到 $\boldsymbol{L}_1 \boldsymbol{D}$ 的次对角线与所给矩阵 $\boldsymbol{A}$ 相同：

① 对角占优阵的定义见 6.1 节.

$$L_1 D = \begin{bmatrix} d_1 & & & & \mathbf{0} \\ d_1 l_2 & d_2 & & & \\ & d_2 l_3 & d_3 & & \\ & & \ddots & \ddots & \\ \mathbf{0} & & & d_{n-1} l_n & d_n \end{bmatrix} = \begin{bmatrix} d_1 & & & & \mathbf{0} \\ a_2 & d_2 & & & \\ & a_3 & d_3 & & \\ & & \ddots & \ddots & \\ \mathbf{0} & & & a_n & d_n \end{bmatrix}$$

不难导出下列计算公式

$$\begin{aligned} & d_1 = b_1 \\ & u_i = c_i / d_i, \\ & l_{i+1} = a_{i+1} / d_i, \quad i = 1, 2, \cdots, n-1 \\ & d_{i+1} = b_{i+1} - u_i a_{i+1}, \end{aligned}$$

据此可自上而下逐行求出分解阵的元素

$$d_1 \to u_1 \to l_2 \to d_2 \to u_2 \to \cdots \to l_n \to d_n$$

题 5 将题 1 的三对角阵 $\boldsymbol{A}$ 分解为 $\boldsymbol{A} = \boldsymbol{U}_1 \boldsymbol{D} \boldsymbol{L}_1$ 的形式，这里 $\boldsymbol{U}_1$，$\boldsymbol{L}_1$ 分别为单位上二对角阵与单位下二对角阵，$\boldsymbol{D}$ 为对角阵.

解 按矩阵乘法规则展开 $\boldsymbol{A} = \boldsymbol{U}_1 \boldsymbol{D} \boldsymbol{L}_1$，其右端为

$$\begin{bmatrix} 1 & u_1 & & & \mathbf{0} \\ & 1 & u_2 & & \\ & & \ddots & \ddots & \\ & & & 1 & u_{n-1} \\ \mathbf{0} & & & & 1 \end{bmatrix} \begin{bmatrix} d_1 & & & & \mathbf{0} \\ & d_2 & & & \\ & & \ddots & & \\ & & & d_{n-1} & \\ \mathbf{0} & & & & d_n \end{bmatrix} \begin{bmatrix} 1 & & & & \mathbf{0} \\ l_2 & 1 & & & \\ & \ddots & \ddots & & \\ & & l_{n-1} & 1 & \\ \mathbf{0} & & & l_n & 1 \end{bmatrix}$$

不难导出计算公式

$$\begin{aligned} & d_n = b_n \\ & l_i = a_i / d_i, \\ & u_{i-1} = c_{i-1} / d_i, \qquad i = n, n-1, \cdots, 2 \\ & d_{i-1} = b_{i-1} - l_i c_{i-1}, \end{aligned}$$

据此可自下而上逐行求出分解阵的元素

$$d_n \to l_n \to u_{n-1} \to d_{n-1} \to l_{n-1} \to \cdots \to u_1 \to d_1$$

题 6 将矩阵

$$\boldsymbol{A} = \begin{bmatrix} b_1 & 0 & c_1 & 0 & 0 \\ 0 & b_2 & 0 & c_2 & 0 \\ a_3 & 0 & b_3 & 0 & c_3 \\ 0 & a_4 & 0 & b_4 & 0 \\ 0 & 0 & a_5 & 0 & b_5 \end{bmatrix}$$

进行 $A=LU_1$ 分解，这里 L 为下三角阵，U_1 为单位上三角阵，试问分解阵 L,U_1 具有什么样的结构？并具体列出分解公式.

解 试令矩阵分解 $A=LU_1$ 具有形式

$$\begin{bmatrix} b_1 & 0 & c_1 & 0 & 0 \\ 0 & b_2 & 0 & c_2 & 0 \\ a_3 & 0 & b_3 & 0 & c_3 \\ 0 & a_4 & 0 & b_4 & 0 \\ 0 & 0 & a_5 & 0 & b_5 \end{bmatrix} = \begin{bmatrix} d_1 & 0 & 0 & 0 & 0 \\ 0 & d_2 & 0 & 0 & 0 \\ l_3 & 0 & d_3 & 0 & 0 \\ 0 & l_4 & 0 & d_4 & 0 \\ 0 & 0 & l_5 & 0 & d_5 \end{bmatrix} \begin{bmatrix} 1 & 0 & u_1 & 0 & 0 \\ 0 & 1 & 0 & u_2 & 0 \\ 0 & 0 & 1 & 0 & u_3 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

按矩阵乘法规则展开有

$$b_1=d_1, \quad c_1=d_1u_1, \quad b_2=d_2, \quad c_2=d_2u_2$$
$$a_3=l_3, \quad b_3=l_3u_1+d_3, \quad c_3=d_3u_3$$
$$a_4=l_4, \quad b_4=l_4u_2+d_4$$
$$a_5=l_5, \quad b_5=l_5u_3+d_5$$

据此立即得出 l_i,d_i,u_i 的计算公式.

题解 6.2 对称阵的 LL^T 分解

提要 如果所给矩阵 $A=(a_{ij})_{n\times n}$ 是对称阵，则矩阵分解亦具有对称结构. 很自然，人们首先考察 $A=LL^T$ 的分解方式，这里 L 为下三角阵.

按矩阵乘法规则展开 $A=LL^T$：

$$\begin{bmatrix} a_{11} & & & & \\ a_{21} & a_{22} & \text{对} & & \\ a_{31} & a_{32} & a_{33} & \text{称} & \\ \vdots & \vdots & \vdots & \ddots & \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & & & & \mathbf{0} \\ l_{21} & l_{22} & & & \\ l_{31} & l_{32} & l_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{nn} \end{bmatrix} \times \begin{bmatrix} l_{11} & l_{21} & l_{31} & \cdots & l_{n1} \\ & l_{22} & l_{32} & \cdots & l_{n2} \\ & & l_{33} & \cdots & l_{n3} \\ & & & \ddots & \vdots \\ \mathbf{0} & & & & l_{nn} \end{bmatrix}$$

矩阵分解的计算公式为

$$l_{ij}=\left(a_{ij}-\sum_{k=1}^{j-1}l_{ik}l_{jk}\right)/l_{jj}, \quad j=1,2,\cdots,i-1$$

$$l_{ii}=\sqrt{a_{ii}-\sum_{k=1}^{i-1}l_{ik}^2}, \qquad i=1,2,\cdots,n$$

上述方法由于计算公式中含有开方运算而称为**平方根法**.

平方根法由于需要求开方，实用价值不大. 但这种方法揭示了这样的事实：一般形式的对称方程组可以化归为两个互为转置的三角方程组来求解.

事实上，基于矩阵分解 $A=LL^T$，方程组 $Ax=b$ 即 $L(L^Tx)=b$ 可化归为两个三角方程组 $Ly=b$，$L^Tx=y$. 其中下三角方程组 $Ly=b$ 即

$$\begin{cases} l_{11}y_1=b_1 \\ \sum_{j=1}^{i} l_{ij}y_j=b_i, \quad i=2,3,\cdots,n \end{cases}$$

的回代公式为

$$\begin{cases} y_1=b_1/l_{11} \\ y_i=\left(b_i-\sum_{j=1}^{i-1} l_{ij}y_j\right)/l_{ii}, \quad i=2,3,\cdots,n \end{cases}$$

这是个顺序计算 $y_1\to y_2\to\cdots\to y_n$ 的追的过程.

此外，上三角方程组 $L^Tx=y$ 即

$$\begin{cases} \sum_{j=i}^{n} l_{ji}x_j=y_i, \quad i=1,2,\cdots,n-1 \\ l_{nn}x_n=y_n \end{cases}$$

的回代公式为

$$\begin{cases} x_n=y_n/l_{nn} \\ x_i=\left(y_i-\sum_{j=i+1}^{n} l_{ji}x_j\right)/l_{ii}, \quad i=n-1,n-2,\cdots,1 \end{cases}$$

这是个逆序计算 $x_n\to x_{n-1}\to\cdots\to x_1$ 的赶的过程.

题 1 将下列对称三对角阵分解为两个二对角阵的乘积：

$$\begin{bmatrix} 1 & 1 & & & \mathbf{0} \\ 1 & 2 & 1 & & \\ & 1 & 3 & 1 & \\ & & 1 & 4 & 1 \\ \mathbf{0} & & & 1 & 5 \end{bmatrix}=\begin{bmatrix} d_1 & & & & \mathbf{0} \\ l_2 & d_2 & & & \\ & l_3 & d_3 & & \\ & & l_4 & d_4 & \\ \mathbf{0} & & & l_5 & d_5 \end{bmatrix}\times\begin{bmatrix} d_1 & l_2 & & & \mathbf{0} \\ & d_2 & l_3 & & \\ & & d_3 & l_4 & \\ & & & d_4 & l_5 \\ \mathbf{0} & & & & d_5 \end{bmatrix}$$

解 按矩阵乘法规则展开，考虑到对称性，可列出方程

$$1=d_1^2, \quad 1=l_2d_1, \quad 2=l_2^2+d_2^2, \quad 1=l_3d_2$$
$$3=l_3^2+d_3^2, \quad 1=l_4d_3, \quad 4=l_4^2+d_4^2, \quad 1=l_5d_4, \quad 5=l_5^2+d_5^2$$

据此求得

$$d_1=1, \quad l_2=1, \quad d_2=1, \quad l_3=1, \quad d_3=\sqrt{2}$$
$$l_4=\frac{1}{\sqrt{2}}, \quad d_4=\sqrt{\frac{7}{2}}, \quad l_5=\sqrt{\frac{2}{7}}, \quad d_5=\sqrt{\frac{33}{7}}$$

题 2 设 A 为对称阵，且具有形式

$$A=\begin{bmatrix} A_1 & a_1 \\ a_1^T & a_{nn} \end{bmatrix}$$

又 $A_1=L_1D_1L_1^T$，这里 L_1 为单位下三角阵，D_1 为对角阵，试确定 c_1 与 d_n，使成立 $A=LDL^T$，式中

$$L=\begin{bmatrix} L_1 & 0 \\ c_1^T & 1 \end{bmatrix},\quad D=\begin{bmatrix} D_1 & 0 \\ 0 & d_n \end{bmatrix}$$

解 直接展开，有

$$a_1=L_1D_1c_1,$$
$$a_{nn}=d_n+c_1^TD_1c_1$$

由此可解出

$$c_1=D_1^{-1}L_1^{-1}a_1$$
$$d_n=a_{nn}-c_1^TD_1c_1$$

题 3 设

$$A=\begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & a \\ 0 & a & 2 \end{bmatrix}$$

(1) 试问数 a 在什么范围内取值，方能保证 A 为正定阵.

(2) 如果 A 为对称正定阵，则它可以进行 Cholesky 分解 $A=L_1DL_1^T$. 试问这里分解阵具有什么样的结构?

(3) 取 $a=1$ 具体列出 Cholesky 分解的计算公式.

解 (1) 考察 A 的顺序主子式:

$$D_1=2$$
$$D_2=\begin{vmatrix} 2 & 1 \\ 1 & 2 \end{vmatrix}=3$$
$$D_3=\begin{vmatrix} 2 & 1 & 0 \\ 1 & 2 & a \\ 0 & a & 2 \end{vmatrix}=6-2a^2$$

为保证 A 为正定阵，要求其顺序主子式全是正的，为此要求 $D_3=6-2a^2>0$，即要求

$$-\sqrt{3}<a<\sqrt{3}$$

(2) 注意到 A 是个三对角阵，知分解阵 L_1 应为单位下二对角阵.

(3) 当 $a=1$ 时令

$$\begin{bmatrix}2&1&0\\1&2&1\\0&1&2\end{bmatrix}=\begin{bmatrix}1&&0\\l_2&1&\\0&l_3&1\end{bmatrix}\begin{bmatrix}d_1&&0\\&d_2&\\0&&d_3\end{bmatrix}\begin{bmatrix}1&l_2&0\\&1&l_3\\0&&1\end{bmatrix}$$

$$=\begin{bmatrix}1&&0\\l_2&1&\\0&l_3&1\end{bmatrix}\begin{bmatrix}d_1&d_1d_2&0\\&d_2&d_2l_3\\0&&d_3\end{bmatrix}$$

按矩阵乘法规则展开,易得

$$d_1=2,\quad l_2=\frac{1}{2},\quad d_2=\frac{3}{2}$$

$$l_3=\frac{2}{3},\quad d_3=\frac{4}{3}$$

习 题 六

1. 用追赶法求解下列方程组:

$$\begin{bmatrix}2&-1&&0\\1&3&-2&\\&-1&2&-1\\0&&-3&5\end{bmatrix}\begin{bmatrix}x_1\\x_2\\x_3\\x_4\end{bmatrix}=\begin{bmatrix}6\\1\\0\\1\end{bmatrix}$$

2. 设

$$A=\begin{bmatrix}a_1&1&&&0\\1&a_2&1&&\\&\ddots&\ddots&\ddots&\\&&1&a_{n-1}&1\\0&&&1&a_n\end{bmatrix}$$

试导出形如 $A=L_1DL_1^T$ 的分解公式,这里 L_1 为单位下二对角阵,D 为对角阵.

3. 试将下列三对角阵 A 分解为 $L_1DL_1^T$ 的形式,其中 L_1 为单位下二对角阵,D 为对角阵:

$$A=\begin{bmatrix}1&1&&&0\\1&2&1&&\\&1&3&1&\\&&1&4&1\\0&&&1&5\end{bmatrix}$$

4. 证明:若6.1节的方程组(4)按下述意义为对角占优:

$$\begin{cases}|b_1|>|c_1|,\\|b_i|\geqslant|a_i|+|c_i|,\quad a_ic_i\neq0,\quad i=2,3,\cdots,n-1\\|b_n|>|a_n|\end{cases}$$

则定理 1 的论断依然正确.

5. 将下列矩阵 $\boldsymbol{A}$ 分解为 $\boldsymbol{LL}^{\mathrm{T}}$,这里 $\boldsymbol{L}$ 为对角线元素为正的下三角阵:

$$\boldsymbol{A}=\begin{bmatrix}3 & 2 & 3\\ 2 & 2 & 0\\ 3 & 0 & 12\end{bmatrix}$$

6. 用 Cholesky 方法求解方程组

$$\begin{cases}4x_1-2x_2+4x_3=8.7\\ -2x_1+17x_2+10x_3=13.7\\ 4x_1+10x_2+9x_3=-0.7\end{cases}$$

7. 分别应用 Crout 分解和 Doolittle 分解求解如下方程组:

$$\begin{bmatrix}5 & 7 & 9 & 10\\ 6 & 8 & 10 & 9\\ 7 & 10 & 8 & 7\\ 5 & 7 & 6 & 5\end{bmatrix}\begin{bmatrix}x_1\\ x_2\\ x_3\\ x_4\end{bmatrix}=\begin{bmatrix}1\\ 1\\ 1\\ 1\end{bmatrix}$$

8. 用 Gauss 消去法解下列方程组:

(1) $\begin{cases}x_1-2x_2=3\\ 2x_1+x_2=4\end{cases}$　　　(2) $\begin{cases}3x_1-x_2+2x_3=-3\\ x_1+x_2+x_3=-4\\ 2x_1+x_2-x_3=-3\end{cases}$

9. 用主元素消去法解下列方程组:

(1) $\begin{cases}x_1-x_2+x_3=-4\\ 5x_1-4x_2+3x_3=-12\\ 2x_1+x_2+x_3=11\end{cases}$　　　(2) $\begin{cases}2x_1+3x_2+5x_3=5\\ 3x_1+4x_2+7x_3=6\\ x_1+3x_2+3x_3=5\end{cases}$

习题参考答案

习　题　一

2. $V(x_0,x_1,\cdots,x_{n-1},x)$有 n 个零点 $x_0,x_1,\cdots,x_{n-1}$，且其首项系数为 $V(x_0,x_1,\cdots,x_{n-1})$；据此反复递推

4. 令 $p(x)=ax^2+b$，用待定系数法

5. $p(x)=x^3-x^2+1$；$p(x)=x^2-1$

6. $p(x)=x^2+1$

7,8. 均用余项校正法

9. $p(x)=5x^4-4x^3+2x^2-2x-1$

习　题　二

2. 2 次

3. $A_0=A_2=\dfrac{h}{3}$，$A_1=\dfrac{4}{3}h$，3 次；$A_0=\dfrac{3}{4}$，$x_0=\dfrac{2}{3}$，1 次

4. 3 次

5. $A_0=A_2=\dfrac{1}{6}$，$A_1=\dfrac{2}{3}$

6. 作变换 $x=t-2$

7. $\omega=\dfrac{1}{3}$

8. 1；1；2

习　题　三

2. $y_n=\dfrac{1}{2}ax_nx_{n+1}+bx_n$

3. 对 $y_{n+1}-y_n=\dfrac{h}{2}[ax_n+b+(ax_{n+1}+b)]$两端累加求和

4. $y_{n+1}=-4y_n+5y_{n-1}+2h(2y'_n+y'_{n-1})$

5. 应用极限$\lim\limits_{x\to 0}(1+x)^{\frac{1}{x}}=e$

7. $a=\dfrac{3}{2}$，$b=-\dfrac{1}{2}$

8. $y_{n+1}=\dfrac{1}{2}(y_n+y_{n-1})+\dfrac{h}{4}(7y'_n-y'_{n-1})$，二阶方法

9. 三阶格式是 $y_{n+1}=2y_n-y_{n-1}+\frac{h}{2}(y'_{n+1}-y'_{n-1})$

习　题　四

4. $\varphi(x)=\frac{1}{2}x(3-ax^2)$

6. $\varphi(x)=\frac{1}{1+x}$

7. 发散；收敛；发散；收敛

习　题　五

1. 0.600，0.200

2. 对角占优

3. 发散

4. 加工成对角占优的同解方程组

7. (1) Jacobi 迭代收敛，1.995 666，－5.002 442，3.001 062；Gauss-Seidel 迭代发散
(2) Jacobi 迭代发散，Gauss-Seidel 迭代收敛，1，2，－4

8. 1.500 00，3.333 33，－2.166 67

习　题　六

1. 5，4，3，2

3. $\boldsymbol{L}$ 的次对角元素 1，1，1/2，2/7；$\boldsymbol{D}$ 的主对线元素 1，1，2，7/2，33/7

5. $\boldsymbol{L}$ 各列元素$\sqrt{3}$，$2/\sqrt{3}$，$\sqrt{3}$；$\sqrt{2/3}$，$-\sqrt{6}$；$\sqrt{3}$

6. －5.156 250，　3.182 500，5.750 000

7. Crout 分解

$$\boldsymbol{L}=\begin{bmatrix}5.0 & 0 & 0 & 0\\ 6.0 & -0.4 & 0 & 0\\ 7.0 & 0.2 & 5.0 & 0\\ 5.0 & 0 & -3.0 & 0.1\end{bmatrix},\quad \boldsymbol{U}_1=\begin{bmatrix}1.0 & 1.4 & 1.8 & 2.0\\ 0 & 1.0 & 2.0 & 7.5\\ 0 & 0 & 1.0 & 1.7\\ 0 & 0 & 0 & 1.0\end{bmatrix}$$

解为 20，－12，－5，3

8. 2.2，－0.4；－1，－2，　1

9. 3，6，　1；　4，1，2

附录　MATLAB 文件汇集

作为附录，这里汇集了一些常用算法的 MATLAB 文件及算例，仅供读者参考．本附录中所有文件和算例都经过调试而成，尽管如此，或许仍有不当之处，望指正．这些文件主要包括：

第一章　插值方法

文件 1.1　Lagrange 插值
文件 1.2　逐步插值
文件 1.3　分段三次 Hermite 插值
文件 1.4　分段三次样条插值

第二章　数值积分

文件 2.1　Simpson 公式
文件 2.2　变步长梯形法
文件 2.3　Romberg 加速算法
文件 2.4　三点 Gauss 公式

第三章　常微分方程的差分方法

文件 3.1　改进的 Euler 方法
文件 3.2　四阶 Runge-Kutta 方法
文件 3.3　二阶 Adams 预报校正系统
文件 3.4　改进的四阶 Adams 预报校正系统

第四章　方程求根

文件 4.1　二分法
文件 4.2　开方法
文件 4.3　Newton 下山法
文件 4.4　快速弦截法

第五章　线性方程组的迭代法

文件 5.1　Jacobi 迭代
文件 5.2　Gauss-Seidel 迭代
文件 5.3　超松弛迭代
文件 5.4　对称超松弛迭代

第六章　线性方程组的直接法

文件 6.1　追赶法

文件 6.2 Cholesky 方法

文件 6.3 矩阵分解方法

文件 6.4 消去法

第一章　插值方法

文件 1.1　Lagrange 插值

文件功能：计算 Lagrange 插值多项式在 $x = x0$ 处的值.

文件名：Lagrange_eval.m

MATLAB 文件：

```
function [y0,N] = Lagrange_eval(X,Y,x0)
%X,Y 是已知插值点坐标
%x0 是插值点
%y0 是 Lagrange 插值多项式在 x0 处的值
%N 是 Lagrange 插值函数的权系数
m = length(X);
N = zeros(m,1);
y0 = 0;
for i = 1:m
    N(i) = 1;
    for j = 1:m
        if j~ = i
            N(i) = N(i) * (x0-X(j))/(X(i)-X(j));
        end
    end
    y0 = y0 + Y(i) * N(i);
end
```

算例：已知 $f(x) = \ln x$ 的数值表如附表 1.1 所示.

附表 1.1

x	0.4	0.5	0.6	0.7	0.8
$\ln x$	−0.916 291	−0.693 147	−0.510 826	−0.356 675	−0.223 144

用线性插值、二次插值、三次插值计算 ln 0.54 的近似值.

解答：

令 X=[0.5,0.6];Y=[−0.693 147,−0.510 826];x0=0.54;运行[y0,N] = Lagrange_eval(X,Y,x0),结果为：

y0 = −6.202 185 999 999 998e−001

这是用线性插值所求得的结果.

若令 X=[0.4,0.5,0.6];Y=[−0.916 291,−0.693 147,−0.510 826];x0=0.54;运行

[y0,N] = Lagrange_eval(X,Y,x0),结果为:

$$y0 = -6.153\ 198\ 399\ 999\ 997e-001$$

这是用二次插值所求得的结果.

若令 X = [0.4, 0.5, 0.6, 0.7]; Y = [-0.916 291, -0.693 147, -0.510 826, -0.356 675]; x0 = 0.54; 运行[y0,N] = Lagrange_eval(X,Y,x0),结果为:

$$y0 = -6.160\ 284\ 079\ 999\ 997e-001$$

这是用三次插值所求得的结果.

而准确值 ln 0.54 = -6.161 861 394 238 170e-001,由此可知三次插值的结果最为准确.

文件 1.2 逐步插值

文件功能: 计算逐步插值多项式在 x = x0 处的值.

文件名: Neville_eval.m

MATLAB 文件:

```
function y0 = Neville_eval(X,Y,x0)
%X,Y 是已知插值点坐标
%x0 是插值点
%y0 是 Neville 逐步插值多项式在 x0 处的值
m = length(X);
P = zeros(m,1);
P1 = zeros(m,1);
P = Y;
for i = 1:m
    P1 = P;
    k = 1;
    for j = i+1:m
        k = k+1;
        P(j) = P1(j-1)+(P1(j)-P1(j-1))*(x0-X(k-1))/(X(j)-X(k-1));
    end
    if abs(P(m)-P(m-1))<10^-6;
        y0 = P(m);
        return;
    end
end
y0 = P(m);
```

因 Neville 逐步插值同 Lagrange 插值所得结果基本上是相同的,故在这里略去算例.但是 Neville逐步插值应该比 Lagrange 插值更优越,对于 Lagrange 插值来说,临时增加一个节点,则需全部重新计算;而 Neville 逐步插值可在原计算基础上进行某种修正即可,这种处理方式使

于实际应用.

文件 1.3　分段三次 Hermite 插值

文件功能：利用分段三次 Hermite 插值计算插值点处的函数近似值.

文件名：Hermite _ interp.m

MATLAB 文件：

```
function y0 = Hermite _ interp(X,Y,DY,x0)
%X,Y 是已知插值点向量序列
%DY 是插值点处的导数值
%x0 是插值点横坐标
%y0 是待求的分段三次 Hermite 插值多项式在 x0 处的值
%N 表示向量长度
N = length(X);
for i = 1:N
    if x0 > = X(i)& x0< = X(i+1)
        k = i; break;
    end
end
a1 = x0 - X(k+1);
a2 = x0 - X(k);
a3 = X(k) - X(k+1);
y0 = (a1/a3)^2 * (1 - 2 * a2/a3) * Y(k) + ( - a2/a3)^2 * (1 + 2 * a1/a3) * Y(k+1) + …
    (a1/a3)^2 * a2 * DY(k) + ( - a2/a3)^2 * a1 * DY(k+1);
```

算例：已知 $\ln x$ 在 $x_1 = 0.30, x_2 = 0.40, x_3 = 0.50, x_4 = 0.60$ 处的函数值及导数值，使用分段三次 Hermite 插值公式计算 $\ln x$ 在 $x = 0.45$ 处的函数值.

解答：

令 X = [0.30,0.40,0.50,0.60]；Y = ln X；DY = 1/X；x0 = 0.45；运行 y0 = Hermite _ interp (X,Y,DY,x0).结果为：

y0 = −7.984 689 562 170 502e−001

其准确值为：−7.985 076 962 177 716e−001.由此可知，用分段三次 Hermite 插值求解的精度还是比较高的.

文件 1.4　分段三次样条插值

文件功能：计算在插值点处的函数值，并用来拟合曲线.

文件名：Spline _ interp.m

MATLAB 文件：

```
function [y0,C] = Spline _ interp(X,Y,s0,sN,x0)
%X,Y 是已知插值点坐标
```

```
%s0,sN 是两端点的一次导数值
%x0 是插值点
%y0 是三次样条函数在 x0 处的值
%C 是分段三次样条函数的系数
N = length(X);
C = zeros(4,N-1);      h = zeros(1,N-1);
mu = zeros(1,N-1);     lmt = zeros(1,N-1);
d = zeros(1,N);               %d 表示右端函数值
h = X(1,2:N) - X(1,1:N-1);
mu(1,N-1) = 1;        lmt(1,1) = 1;
mu(1,1:N-2) = h(1,1:N-2)./(h(1,1:N-2) + h(1,2:N-1));
lmt(1,2:N-1) = h(1,2:N-1)./(h(1,1:N-2) + h(1,2:N-1));
d(1,1) = 6 * ((Y(1,2) - Y(1,1))/h(1,1) - s0)/h(1,1);
d(1,N) = 6 * (sN - (Y(1,N) - Y(1,N-1))/h(1,N-1))/h(1,N-1);
d(1,2:N-1) = 6 * ((Y(1,3:N) - Y(1,2:N-1))./h(1,2:N-1) - ...
    (Y(1,2:N-1) - Y(1,1:N-2))./h(1,1:N-2))./(h(1,1:N-2) + h(1,2:N-1));
%追赶法解三对角方程组
bit = zeros(1,N-1);
bit(1,1) = lmt(1,1)/2;
for i = 2:N-1
    bit(1,i) = lmt(1,i)/(2-mu(1,i-1) * bit(1,i-1));
end
y = zeros(1,N);
y(1,1) = d(1,1)/2;
for i = 2:N
    y(1,i) = (d(1,i) - mu(1,i-1) * y(1,i-1))/(2 - mu(1,i-1) * bit(1,i-1));
end
x = zeros(1,N);
x(1,N) = y(1,N);
for i = N-1:-1:1
    x(1,i) = y(1,i) - bit(1,i) * x(1,i+1);
end

v = zeros(1,N-1);
v(1,1:N-1) = (Y(1,2:N) - Y(1,1:N-1))./h(1,1:N-1);
C(4,:) = Y(1,1:N-1);
C(3,:) = v - h .* (2 * x(1,1:N-1) + x(1,2:N))/6;
C(2,:) = x(1,1:N-1)/2;
```

```
C(1,:) = (x(1,2:N) - x(1,1:N-1))/(6 * h);
if nargin<5
    y0 = 0;
else
    for j = 1:N-1
        if x0> = X(1,j)& x0<X(1,j+1)
            omg = x0 - X(1,j);
            y0 = ((C(4,j) * omg + C(3,j)) * omg + C(2,j)) * omg + C(1,j);
        end
    end
end
```

算例：给定数据表如附表 1.2 所示.

附表 1.2

x_i	0.25	0.30	0.39	0.45	0.53
y_i	0.500 0	0.547 7	0.624 5	0.670 8	0.728 0

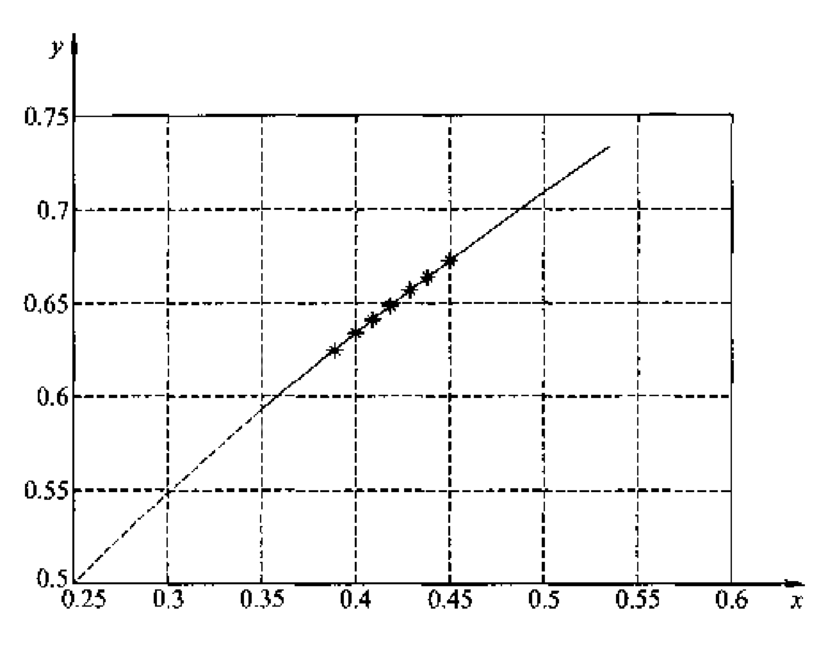
试求三次样条插值函数 $S(x)$. 并满足条件：$S'(0.25)=1.000\ 0, S'(0.53)=0.686\ 8$.

解答：

令 X=[0.25,0.30,0.39,0.45,0.53];Y=[0.500 0,0.547 7,0.624 5,0.670 8,0.728 0];s0=1.000 0;sN=0.686 8;运行

```
[y0,C] = Spline_interp(X,Y,s0,sN,x0);
plot(0.25:0.01:0.30,polyval(C(:,1),0:0.01:0.05)'r-.');
hold on
plot(0.30:0.01:0.39,polyval(C(:,2),0:0.01:0.09),'b');
plot(0.39:0.01:0.45,polyval(C(:,3),0:0.01:0.06),'k-*');
plot(0.45:0.01:0.53,polyval(C(:,4),0:0.01:0.08));
```

得到如附图 1.1 所示的三次样条插值函数 $S(x)$的曲线图.



附图 1.1

第二章　数值积分

文件 2.1　复化 Simpson 公式

文件功能：利用复化 Simpson 公式求被积函数 $f(x)$ 在给定区间上的积分值.

文件名：FSimpson.m

MATLAB 文件：

```
function S=FSimpson(f,a,b,N)
%f表示被积函数句柄
%a,b表示被积区间[a,b]的端点
%N表示区间个数
%S是用复化Simpson公式求得的积分值
h=(b-a)/N;
fa=feval(f,a);
fb=feval(f,b);
S=fb+fa;
x=a;
for i=1:N
    x=x+h/2;
    fx=feval(f,x);
    S=S+4*fx;
    x=x+h/2;
    fx=feval(f,x);
    S=S+2*fx;
end
S=h*S/6;
```

算例：利用复化 Simpson 公式计算积分 $S=\int_0^1 \frac{x}{4+x^2}dx$.

解答：

令 f=@f1;a=0;b=1;运行

S=FSimpson(f,a,b,N);

若 N=16;结果为:　　S=1.157 384 446 683 415e-001

若 N=64;结果为:　　S=1.126 134 423 329 238e-001

若 N=256;结果为:　　S=1.118 321 923 238 073e-001

而积分 $S=\int_0^1 \frac{x}{4+x^2}dx$ 的准确值为:

$S=1.115\,717\,756\,571\,049e-001$

由此可知,用复化 Simpson 公式求积,区间越小,求得的精度越高,但整体效果并不是太好,有待进一步改进.

注:这里,被积函数以如下所示的文件形式表达(后同).

```
function f=f1(x)
f=x/(4+x^2);
```

文件 2.2 变步长梯形法

文件功能:利用变步长梯形法计算被积函数 $f(x)$ 在给定区间上的积分值.

文件名:bbct.*m*

MATLAB 文件:

```
function [T,n]=bbct(f,a,b,eps)
%f 表示被积函数句柄
%a,b 表示被积区间[a,b]的端点
%eps 表示精度
%T 是用变步长梯形法求得的积分值
%n 表示二分区间的次数
h=b-a;
fa=feval(f,a);
fb=feval(f,b);
T1=h*(fa+fb)/2;
T2=T1/2+h*feval(f,a+h/2)/2;
n=1;
%按变步长梯形法求积分值;
while abs(T2-T1)>=eps
    h=h/2;
    T1=T2;
    S=0;
    x=a+h/2;
    while x<b
        fx=feval(f,x);
        S=S+fx;
        x=x+h;
    end
    T2=T1/2+S*h/2;
    n=n+1;
end
T=T2;
```

算例:利用变步长梯形法计算积分 $T=\int_0^1 \frac{x}{4+x^2}\mathrm{d}x$.

解答：

令 f=@f1;a=0;b=1;运行

[T,n]=bbct(f,a,b,eps);

若 eps=0.001;结果为:T=1.114 023 545 295 480e-001　n=3;

若 eps=0.000 1;结果为:T=1.115 611 956 442 211e-001　n=5;

若 eps=0.000 01;结果为:T=1.115 691 307 637 255e-001　n=6;

由此可知,变步长梯形法是根据精度指标确定步长的大小,这就避免了步长太小(计算量增加)和步长太大(精度不够)的缺点.

文件 2.3 Romberg 加速算法

文件功能： 利用 Romberg 加速算法计算被积函数 $f(x)$在给定区间上的积分值.

文件名： Romberg.m

MATLAB 文件：

```
function [quad, R]=Romberg(f,a,b,eps)
%f 表示被积函数句柄
%a,b 表示被积区间[a,b]的端点
%eps 表示精度
%quad 是用 Romberg 加速算法求得的积分值
%R 为 Romberg 表
%err 表示误差的估计
h=b-a;
R(1,1)=h*(feval(f,a)+feval(f,b))/2;
M=1;J=0;err=1;
while err>eps
    J=J+1;
    h=h/2;
    S=0;
    for p=1:M
        x=a+h*(2*p-1);
        S=S+feval(f,x);
    end
    R(J+1,1)=R(J,1)/2+h*S;
    M=2*M;
    for k=1:J
        R(J+1,k+1)=R(J+1,k)+(R(J+1,k)-R(J,k))/(4^k-1);
    end
    err=abs(R(J+1,J)-R(J+1,J+1));
end
```

```
quad = R(J+1,J+1);
```

算例：利用 Romberg 加速算法计算积分 $R=\int_0^1 \frac{x}{4+x^2}dx$.

解答：

令 f = @f1; a = 0; b = 1; 运行

[quad,R] = Romberg(f,a,b,eps);

若 eps = 10^-4,所得到的 Romberg 表如附表 2.1 所示.

附表 2.1

0.100 000 000 000 00	0	0
0.108 823 529 411 76	0.111 764 705 882 35	0
0.110 892 270 501 46	0.111 581 850 864 69	0.111 569 660 530 18

此时,所求得的积分值为:quad = 0.111 569 660 530 18.

若 eps = 10^-7,所得到的 Romberg 表如附表 2.2 所示.

附表 2.2

0.100 000 000 000 00	0	0	0
0.108 823 529 411 76	0.111 764 705 882 35	0	0
0.110 892 270 501 46	0.111 581 850 864 69	0.111 569 660 530 18	0
0.111 402 354 529 55	0.111 572 382 538 91	0.111 571 751 317 19	0.111 571 784 504 29

此时,所求得的积分值为:quad = 0.111 571 784 504 29.

同准确值 0.111 571 775 657 104 相比,Romberg 加速算法经过较少的计算就能达到很高的精度,加速效果是很显著的.

文件 2.4 三点 Gauss 公式

文件功能：利用三点 Gauss 公式计算被积函数 $f(x)$在给定区间上的积分值.

文件名：TGauss.m

MATLAB 文件：

```
function G = TGauss(f,a,b)
%f 表示被积函数句柄
%a,b 表示被积区间[a,b]的端点
%G 是用三点 Gauss 公式求得的积分值
x1 = (a+b)/2 - sqrt(3/5) * (b-a)/2;
x2 = (a+b)/2 + sqrt(3/5) * (b-a)/2;
G = (b-a) * (5 * feval(f,x1)/9 + 8 * feval(f,(a+b)/2)/9 + 5 * feval(f,x2)/9)/2;
```

算例：利用三点 Gauss 公式计算积分 $R=\int_0^1 \frac{x}{4+x^2}\mathrm{d}x$.

解答：

令 f=@f1;a=0;b=1;运行

G = TGauss(f,a,b);

所得结果为： G=1.115 738 330 730 522e-001.

同文件 2.1 相比，用三点 Gauss 公式求得的积分值，比复化 Simpson 公式的精度还高，由此可以推出：求积节点的选取对数值求积精度的影响是非常大的

第三章　常微分方程的差分方法

文件 3.1　改进的 Euler 方法

文件功能：用改进的 Euler 法求解常微分方程.

文件名：MendEuler.m

MATLAB 文件：

```
function E = MendEuler(f,a,b,N,ya)
%f 是微分方程右端函数句柄
%a,b 是自变量的取值区间[a,b]的端点
%N 是区间等分的个数
%ya 表初值 y(a)
%E = [x',y']是自变量 X 和解 Y 所组成的矩阵
h = (b - a)/N;
y = zeros(1,N + 1);
x = zeros(1,N + 1);
y(1) = ya;
x = a:h:b;
for i = 1:N
    y1 = y(i) + h * feval(f,x(i),y(i));
    y2 = y(i) + h * feval(f,x(i+1),y1);
    y(i+1) = (y1 + y2)/2;
end
E = [x',y'];
```

算例：对于微分方程 $\frac{dy}{dx} = x^2 - y, y(0) = 1, 0 < x < 1$，用改进的 Euler 方法求解.

解答：

对于右端函数，以文件的形式表达如下(后同)：

```
function z = f2(x,y)
z = x^2 - y;
```

为衡量数值解的精度，我们求出该方程的解析解为 $y = -e^{-x} + x^2 - 2x + 2$. 在此也以文件的形式表示如下：

```
function y = solvef2(x)
y = - exp( - x) + x^2 - 2 * x + 2;
```

令 f = @f2;a = 0;b = 1;N = 10;ya = 1;运行

```
E = MendEuler(f,a,b,N,ya);
y = solvef2(a:(b - a)/N:b);
m = [E,y']
```

则 m 为：

m =

0	1.000 000 000 000 00	1.000 000 000 000 00
0.100 000 000 000 00	0.905 500 000 000 00	0.905 162 581 964 04
0.200 000 000 000 00	0.821 927 500 000 00	0.821 269 246 922 02
0.300 000 000 000 00	0.750 144 387 500 00	0.749 181 779 318 28
0.400 000 000 000 00	0.690 930 670 687 50	0.689 679 953 964 36
0.500 000 000 000 00	0.644 992 256 972 19	0.643 469 340 287 37
0.600 000 000 000 00	0.612 967 992 559 83	0.611 188 363 905 97
0.700 000 000 000 00	0.595 436 033 266 65	0.593 414 696 208 59
0.800 000 000 000 00	0.592 919 610 106 31	0.590 671 035 882 78
0.900 000 000 000 00	0.605 892 247 146 21	0.603 430 340 259 40
1.000 000 000 000 00	0.634 782 483 667 32	0.632 120 558 828 56

其中，第 1 列为离散节点值，第 2 列是用改进的 Euler 法求得的解，第 3 列是准确解．根据计算结果可知，改进的 Euler 法求解有一定的准确性，但精度不高，有待进一步改进．

文件 3.2　四阶 Runge-Kutta 方法

文件功能：用四阶 Runge-Kutta 法求解常微分方程．

文件名：Rungkuta4.m

MATLAB 文件：

```
function R = Rungkuta4(f,a,b,N,ya)
%f 是微分方程右端函数句柄
%a,b 是自变量的取值区间[a,b]的端点
%N 是区间等分的个数
%ya 表初值 y(a)
%R = [x',y']是自变量 X 和解 Y 所组成的矩阵
h = (b - a)/N;
x = zeros(1,N + 1);
y = zeros(1,N + 1);
x = a:h:b;
y(1) = ya;
for i = 1:N
    k1 = feval(f,x(i),y(i));
    k2 = feval(f,x(i) + h/2,y(i) + (h/2) * k1);
    k3 = feval(f,x(i) + h/2,y(i) + (h/2) * k2);
    k4 = feval(f,x(i) + h,y(i) + h * k3);
    y(i + 1) = y(i) + (h/6) * (k1 + 2 * k2 + 2 * k3 + k4);
end
```

```
R=[x',y'];
```

算例：对于微分方程 $\frac{dy}{dx}=x^2-y$，$y(0)=1$，$0<x<1$，用四阶 Runge-Kutta 方法求解.

解答：

令 f=@f2;a=0;b=1,N=10;ya=1;运行

```
R=Rungkuta4(f,a,b,N,ya);
y=solvef2(a:(b-a)/N:b);
m=[R,y']
```

则 m 为：

m=

0	1.000 000 000 000 00	1.000 000 000 000 00
0.100 000 000 000 00	0.905 162 708 333 33	0.905 162 581 964 04
0.200 000 000 000 00	0.821 269 495 434 90	0.821 269 246 922 02
0.300 000 000 000 00	0.749 182 145 408 91	0.749 181 779 318 28
0.400 000 000 000 00	0.689 680 432 829 76	0.689 679 953 964 36
0.500 000 000 000 00	0.643 469 926 973 94	0.643 469 340 287 37
0.600 000 000 000 00	0.611 189 053 381 61	0.611 188 363 905 97
0.700 000 000 000 00	0.593 415 483 422 52	0.593 414 696 208 59
0.800 000 000 000 00	0.590 671 915 814 66	0.590 671 035 882 78
0.900 000 000 000 00	0.603 431 307 959 28	0.603 430 340 259 40
1.000 000 000 000 00	0.632 121 609 448 93	0.632 120 558 828 56

其中，三列数据分别为离散节点值、四阶 Runge-Kutta 法求得的解、准确解，根据计算结果可知，与改进的 Euler 法相比，四阶 Runge-Kutta 法的精度是非常高的.

文件 3.3 二阶 Adams 预报校正系统

文件功能：用二阶 Adams 预报校正系统求解常微分方程.

文件名：Adams2PC.m

MATLAB 文件：

```
function A=Adams2PC(f,a,b,N,ya)
%f是微分方程右端函数句柄
%a,b是自变量的取值区间[a,b]的端点
%N是区间等分的个数
%ya表初值y(a)
%A=[x',y']是自变量X和解Y所组成的矩阵
h=(b-a)/N;
x=zeros(1,N+1);
y=zeros(1,N+1);
x=a:h:b;
```

```
y(1) = ya;
for i = 1:N
    if i− = 1
        y1 = y(i) + h * feval(f,x(i),y(i));
        y2 = y(i) + h * feval(f,x(i+1),y1);
        y(i+1) = (y1 + y2)/2;
        dy1 = feval(f,x(i),y(i));
        dy2 = feval(f,x(i+1),y(i+1));
    else
        y(i+1) = y(i) + h * (3 * dy2 − dy1)/2;
        P = feval(f,x(i+1),y(i+1));
        y(i+1) = y(i) + h * (P + dy2)/2;
        dy1 = dy2;
        dy2 = feval(f,x(i+1),y(i+1));
    end
end
A = [x',y'];
```

文件 3.4　改进的四阶 Adams 预报校正系统

文件功能：用改进的四阶 Adams 预报校正系统求解常微分方程.

文件名：CAdams4PC.m

MATLAB 文件：

```
function A = CAdams4PC(f,a,b,N,ya)
%f 是微分方程右端函数句柄
%a,b 是自变量的取值区间[a,b]的端点
%N 是区间等分的个数
%ya 表初值 y(a)
%A = [x',y']是自变量 X 和解 Y 所组成的矩阵
if N < 4
    break;
end
h = (b − a)/N;
x = zeros(1,N+1);
y = zeros(1,N+1);
x = a:h:b;
y(1) = ya;
F = zeros(1,4);
for i = 1:N
```

```
    if i < 4                    %用四阶 Runge-Kutta 法求初始解
        k1 = feval(f,x(i),y(i));
        k2 = feval(f,x(i) + h/2,y(i) + (h/2) * k1);
        k3 = feval(f,x(i) + h/2,y(i) + (h/2) * k2);
        k4 = feval(f,x(i) + h,y(i) + h * k3);
        y(i+1) = y(i) + (h/6) * (k1 + 2 * k2 + 2 * k3 + k4);
    elseif i = = 4
          F = feval(f,x(i-3:i),y(i-3:i));
          py = y(i) + (h/24) * (F * [-9,37,-59,55]');        %预报
          p = feval(f,x(i+1),py);
          F = [F(2)F(3)F(4)p];
          y(i+1) = y(i) + (h/24) * (F * [1,-5,19,9]');       %校正
          p = py;c = y(i+1);
    else
          F = feval(f,x(i-3:i),y(i-3:i));
          py = y(i) + (h/24) * (F * [-9,37,-59,55]');        %预报
          my = py - 251 * (p - c)/270;                        %改进
          m = feval(f,x(i+1),my);
          F = [F(2) F(3) F(4) m];
          cy = y(i) + (h/24) * (F * [1,-5,19,9]');           %校正
          y(i+1) = cy + 19 * (py - cy)/270;                  %改进
          p = py;c = cy;
    end
end
A = [x',y'];
```

附：四阶 Adams 预报校正系统的程序如下：

```
function A = Adams4PC(f,a,b,N,ya)
%f是微分方程右端函数句柄
%a,b是自变量的取值区间[a,b]的端点
%N是区间等分的个数
%ya表初值y(a)
%A=[x',y']是自变量X和解Y所组成的矩阵
if N < 4
    break;
end
h = (b-a)/N;
x = zeros(1, N+1);
y = zeros(1, N+1);
```

```
x = a:h:b;
y(1) = ya;
F = zeros(1, 4);
for i = 1:N
    if i < 4                          % 用四阶 Runge-Kutta 法求初始解
        k1 = feval(f, x(i), y(i));
        k2 = feval(f, x(i) + h/2, y(i) + (h/2) * k1);
        k3 = feval(f, x(i) + h/2, y(i) + (h/2) * k2);
        k4 = feval(f, x(i) + h, y(i) + h * k3);
        y(i + 1) = y(i) + (h/6) * (k1 + 2 * k2 + 2 * k3 + k4);
    else
        F = feval(f, x(i-3:i), y(i-3:i));
        py = y(i) + (h/24) * (F * [-9, 37, -59, 55]');          % 预报
        p = feval(f, x(i+1), py);
        F = [F(2) F(3) F(4) p];
        y(i+1) = y(i) + (h/24) * (F * [1, -5, 19, 9]');          % 校正
    end
end
A = [x', y']
```

算例:分别用二阶 Adams 预报校正系统、四阶 Adams 预报校正系统和改进的四阶 Adams 预报校正系统求解如下微分方程初值问题:$\frac{dy}{dx} = -y + x + 1, y(0) = 1, 0 < x < 1$.

解答:

对于右端函数,以文件的形式表达如下:

```
function z = f3(x, y)
z = -y + x + 1;
```

为衡量数值解的精度,我们求出该方程的解析解为:$y = e^{-x} + x$.在此也以文件的形式表示如下:

```
function y = solvef3(x)
y = exp(-x) + x;
```

令 f = @f3;a = 0;b = 1;N = 10;ya = 1;运行

```
A2 = Adams2PC(f, a, b, N, ya);
A4 = Adams4PC(f, a, b, N, ya);
CA4 = CAdams4PC(f, a, b, N, ya);
y = solvef3(a:(b-a)/N:b);
m = [A2, A4(:, 2), CA4(:, 2), y'];
```

则 m 为:

m =

0	1.000 000 000 000 00	1.000 000 000 000 00	1.000 000 000 000 00	1.000 000 000 000 00
0.1	1.005 000 000 000 00	1.004 837 500 000 00	1.004 837 500 000 00	1.004 837 418 035 96
0.2	1.018 787 500 000 00	1.018 730 901 406 25	1.018 730 901 406 25	1.018 730 753 077 98
0.3	1.040 787 156 250 00	1.040 818 422 001 18	1.040 818 422 001 18	1.040 818 220 681 72
0.4	1.070 217 375 546 87	1.070 319 918 243 95	1.070 319 918 243 95	1.070 320 046 035 64
0.5	1.106 370 300 418 16	1.106 530 268 410 28	1.106 530 577 178 58	1.106 530 659 712 63
0.6	1.148 605 504 190 62	1.148 811 032 554 09	1.148 811 579 275 88	1.148 811 636 094 03
0.7	1.196 343 569 301 94	1.196 584 531 375 83	1.196 585 269 422 73	1.196 585 303 791 41
0.8	1.249 060 275 381 03	1.249 328 060 447 85	1.249 328 949 777 38	1.249 328 964 117 22
0.9	1.306 281 340 985 03	1.306 568 656 793 14	1.306 569 661 369 40	1.306 569 659 740 60
1.0	1.367 577 666 255 46	1.367 878 366 023 76	1.367 879 455 916 38	1.367 879 441 171 44

其中，这五列数据从左到右依次为离散节点值、二阶 Adams 预报校正系统所求解、四阶 Adams 预报校正系统所求解、改进的四阶 Adams 预报校正系统所求解和准确解，通过计算结果的比较分析可知：改进的四阶 Adams 预报校正系统效果最好，其次是四阶 Adams 预报校正系统，二阶 Adams 预报校正系统效果较差.

第四章 方程求根

文件 4.1 二 分 法

文件功能：用二分法求解非线性方程 $f(x)=0$ 在区间$[a,b]$内的根.

文件名：demimethod.m

MATLAB 文件：

```
function [x, k] = demimethod(a, b, f, emg)
%a, b 表示求解区间[a, b]的端点
%f 表示所求解方程的函数名
%emg 是精度指标
%x 表示所求近似解
%k 表示循环次数
fa = feval(f, a);
fab = feval(f,(a+b)/2);
k = 0;
while abs(b-a)>emg
    if fab = = 0
        x = (a+b)/2;
        return;
    elseif fa * fab<0
        b = (a+b)/2;
    else
        a = (a+b)/2;
    end
    fa = feval(f, a);
    fab = feval(f, (a+b)/2);
    k = k+1;
end
x = (a+b)/2;
```

算例：求方程 $f(x)=\sqrt{x^2+1}-\tan x$ 在区间$[0,\pi/2]$内的实根，使精度达到 10^{-5}.

解答：

编写函数文件 func2.m.

```
function f = func2(x)
f = sqrt(x^2+1) - tan(x);
```

在命令窗口中输入：

```
f = @func2;     [x0, k] = demimethod(0, pi/2, f, 10^-5);
```

所得近似根为 x0 = 9.414 597 361 712 279e-001;k = 18;此时,f = feval(f, x0) = 3.934 451 757 503 510e-006;即非线性方程 $f(x) = \sqrt{x^2+1} - \tan x$ 在 x0 = 9.414 597 361 712 279e-001 点趋于零,故 x0 是[0,π/2]内的近似实根.

二分法的优点是算法简单,且总是收敛的.缺点是收敛速度太慢.一个可取的办法是用二分法为后面的加速算法选出适当的初值.

文件 4.2 开 方 法

文件功能:求实数的开方运算.

文件名:Kaifang.m

MATLAB 文件:

```
function y = Kaifang(a, eps, x0)
%a 是被开方数
%eps 是精度指标
%x0 表初值
%y 是 a 的开方
x(1) = x0;
x(2) = (x(1) + a/x(1))/2;
k = 2;
while abs(x(k) - x(k-1)) > eps
    x(k+1) = (x(k) + a/x(k))/2;
    k = k+1;
end
y = x';
```

算例:用开方运算求$\sqrt{2}$,设取 x0 = 1.

解答:

令 a = 2;eps = 10^-6;x0 = 1;运行

y = Kaifang(a, eps, x0);

得 y =

1.000 000 000 000 00

1.500 000 000 000 00

1.416 666 666 666 67

1.414 215 686 274 51

1.414 213 562 374 69

1.414 213 562 373 09

经过 5 次迭代,即可得到 eps = 10^{-6}的结果 1.414 213 562 373 09.

文件 4.3 Newton 下山法

文件功能:用牛顿下山法求解非线性方程 f(x) = 0 的根.

文件名：Mendnewton.m

MATLAB 文件：

```
function [x,k] = Mendnewton(f, x0, emg)
% 用牛顿下山法求解非线性方程
% f 表示非线性方程
% x0 是迭代初值,此方法是局部收敛,初值选择要恰当
% emg 是精度指标
% k,u 分别表示迭代次数和下山因子
[f1, d1] = feval(f, x0);          % d1 表示非线性方程 f 在 x0 处的导数值,以下类同
k = 1;
x(1) = x0;
x(2) = x(1) - f1/d1;
while abs(f1) > emg               % 控制精度
    u = 1;
    k = k + 1;
    [f1,d1] = feval(f,x(k));
    x(k+1) = x(k) - u * f1/d1;  % 牛顿下山迭代
    [f2,d2] = feval(f,x(k+1));
    while abs(f2) > abs(f1)       % 保证迭代后的函数值比迭代前的函数值小
        u = u/2;
        x(k+1) = x(k) - u * f1/d1;   % 牛顿下山迭代
        [f2,d2] = feval(f,x(k+1));
    end
end
```

算例：用牛顿下山法求方程 $f(x)=\sqrt{x^2+1}-\tan x$ 的根，使精度达到 10^{-6}，初值分别选取为：(1) x0 = -1.2；(2) x0 = 2.0；

解答：

编写函数文件 func3.m。

```
function [f,d] = func3(x)
f = sqrt(x^2 + 1) - tan(x);
d1 = 'sqrt(x^2 + 1) - tan(x)';
d = subs(diff(d1));             % 对函数 f 求一次导数
```

在命令窗口中输入：

```
f = @func3;     [x,k] = Mendnewton(f,x0,10^-6);
```

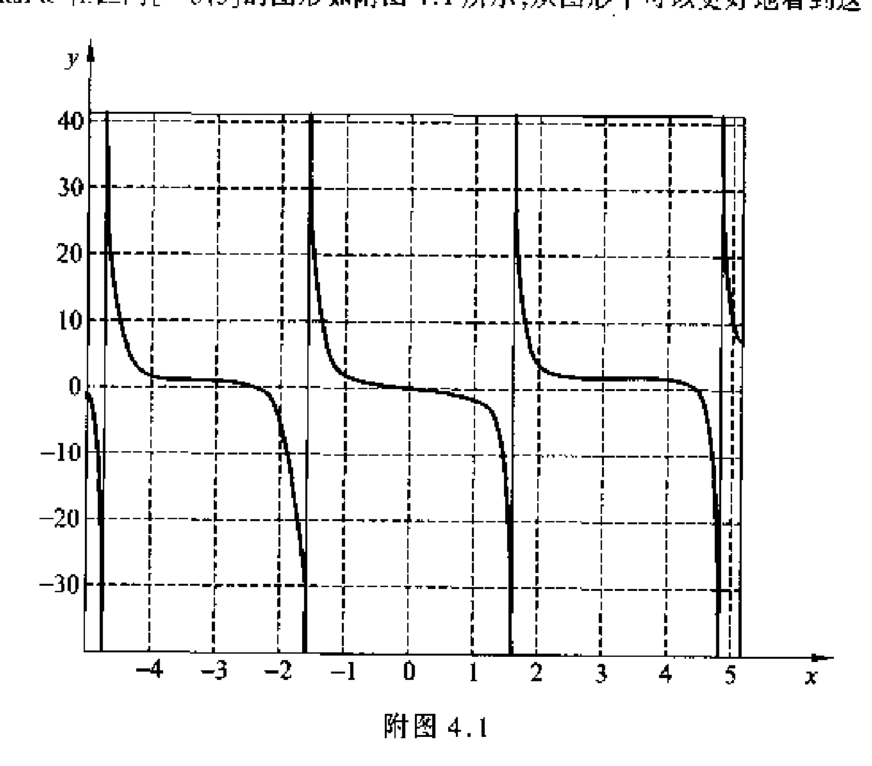
若选初值为 x0 = -1.2；运行结果如下：

迭代次数 k	x 值	f1(k) = feval(f,x(k))值
1	-7.069 047 932 971 935e-001	2.078 789 280 010 764e+000
2	1.942 400 972 108 479e-001	8.219 695 728 408 301e-001
3	1.163 518 073 303 871e+000	-7.838 374 932 606 165e-001
4	1.023 918 977 930 554e+000	-2.113 030 290 935 414e-001
5	9.530 711 345 686 330e-001	-2.606 996 588 743 926e-002
6	9.416 925 081 385 333e-001	-5.085 688 425 774 393e-004
7	9.414 616 152 761 416e-001	-2.012 113 482 496 858e-007
8	9.414 615 238 528 302e-001	-3.153 033 389 935 445e-014

若选初值为 x0 = 2.0;运行结果如下:

迭代次数 k	x 值	f1(k) = feval(f,x(k))值
1	2.905 969 917 234 289e+000	3.313 298 980 588 495e+000
2	3.829 942 435 553 551e+000	3.135 774 879 468 060e+000
3	4.382 754 035 040 099e+000	1.572 413 838 570 136e+000
4	4.474 505 813 415 593e+000	4.607 393 526 441 488e-001
5	4.501 556 126 032 599e+000	-6.131 570 643 391 715e-002
6	4.498 750 820 792 893e+000	8.285 614 610 334 946e-004
7	4.498 711 866 735 406e+000	-1.555 631 969 907 267e-007
8	4.498 711 859 418 998e+000	-1.154 631 945 610 163e-014

牛顿法本身是牛顿下山法的一个特例,因此,我们在这里选取了牛顿下山法,它具有收敛速度快的优点,但是,初值的选取对收敛速度有极大影响.牛顿法本身的局部收敛性决定了必须首先选取恰当的初值.上面的计算机实验表明若选取不同的初值,将收敛到不同的根.方程 $f(x)=\sqrt{x^2+1}-\tan x$ 在区间[-5,5]的图形如附图 4.1 所示,从图形中可以更好地看到这一点.



附图 4.1

文件 4.4 快速弦截法

文件功能:用快速弦截法求解非线性方程 $f(x)=0$ 的根.

文件名:Fast_chord.m

MATLAB 文件:

```
function [x,k]=Fast_chord(f,x1,x2,emg);
%用快速弦截法求解非线性方程的根
%f 表示非线性方程函数
%x1,x2 表示迭代初值
%emg 是精度指标
%k 表示循环次数
k=1;
y1=feval(f,x1);
y2=feval(f,x2);
x(k)=x2-(x2-x1)*y2/(y2-y1);                %用快速弦截法进行迭代求解
y(k)=feval(f,x(k));
k=k+1;
x(k)=x(k-1)-(x(k-1)-x2)*y(k-1)/(y(k-1)-y2);
while abs(x(k)-x(k-1))>emg                  %控制精度
    y(k)=feval(f,x(k));
    x(k+1)=x(k)-(x(k)-x(k-1))*y(k)/(y(k)-y(k-1));
    k=k+1;
end
```

算例:用快速弦截法求解方程 $4\cos x=e^x$ 的根,要求精度为 $\epsilon=10^{-6}$,初值为:$x_1=\pi/4$,$x_2=\pi/2$.

解答:

编写函数文件 func4.m

```
function f=func4(x)
f=exp(x)-4*cos(x)
```

在命令窗口中输入:

```
f=@func4;      [x,k]=Fast_chord(f,pi/4,pi/2,10^-6);
```

运行结果如下:

迭代次数 k	x 值	f1(k)=feval(f,x(k))值
1	8.770 025 972 944 069e-001	-1.541 498 847 256 566e-001
2	8.985 446 421 856 659e-001	-3.497 120 992 034 475e-002
3	9.048 658 349 261 991e-001	4.359 577 241 643 819e-004
4	9.047 880 039 957 831e-001	-1.201 259 723 249 137e-006
5	9.047 882 178 657 145e-001	4.102 540 529 515 864e-011

第五章　线性方程组的迭代法

文件 5.1　Jacobi 迭代

文件功能:用 Jacobi 迭代法求解线性方程组.

文件名:Jacobimethod.m

MATLAB 文件:

```
function [x,k] = Jacobimethod(A,b,x0,N,emg)
%A 是线性方程组的左端矩阵
%b 是右端向量
%x0 是迭代初始值
%N 表示迭代次数上限,若迭代次数大于 N,则迭代失败
%emg 表示控制精度
%用 Jacobi 迭代法求线性方程组 A*x=b 的解
%k 表示迭代次数
%x 表示用迭代法求得的线性方程组的近似解
n = length(A);
x1 = zeros(n,1);   x2 = zeros(n,1);
x1 = x0;   k = 0;
r = max(abs(b - A*x1));
while r>emg
    for i = 1:n
        sum = 0;
        for j = 1:n
            if i~ = j
                sum = sum + A(i,j) * x1(j);
            end
        end
        x2(i) = (b(i) - sum)/A(i,i);
    end
    r = max(abs(x2 - x1));
    x1 = x2;
    k = k + 1;
    if k>N
        disp('迭代失败,返回');
        return;
    end
```

```
end
x = x1;
```

算例:用 Jocobi 迭代法求解方程组:

$$\begin{cases} -4x_1 + x_2 + x_3 + x_4 = 1 \\ x_1 - 4x_2 + x_3 + x_4 = 1 \\ x_1 + x_2 - 4x_3 + x_4 = 1 \\ x_1 + x_2 + x_3 - 4x_4 = 1 \end{cases}$$

其精确解为 x = [-1, -1, -1, -1]'.

解答:

令 A = [-4,1,1,1;1, -4,1,1;1,1, -4,1;1,1,1, -4];b = [1,1,1,1]';x0 = [0,0,0,0]';
在命令窗口中输入:

```
[x,k] = Jacobimethod(A,b,x0,100,10^-5);
```

运行结果为:

```
k = 37
x =
    -9.999 761 621 685 057e-001
    -9.999 761 621 685 057e-001
     9.999 761 621 685 057e-001
    -9.999 761 621 685 057e-001
```

文件 5.2　Gauss-Seidel 迭代

文件功能:用 Gauss-Seidel 迭代法求解线性方程组.

文件名:Gaussmethod.m

MATLAB 文件:

```
function [x,k] = Gaussmethod(A,b,x0,N,emg)
%A 是线性方程组的左端矩阵
%b 是右端向量
%x0 是迭代初始值
%N 表示迭代次数上限,若迭代次数大于 N,则迭代失败
%emg 表示控制精度
%用 Gauss-Seidel 迭代法求线性方程组 A * x = b 的解
%k 表示迭代次数
%x 表示用迭代法求得的线性方程组的近似解
n = length(A);
x1 = zeros(n,1);   x2 = zeros(n,1);
x1 = x0;
r = max(abs(b - A * x1));
k = 0;
```

```
while r>emg
    for i=1:n
        sum=0;
        for j=1:n
            if j>i
                sum=sum+A(i,j)*x1(j);
            elseif j<i
                sum=sum+A(i,j)*x2(j);
            end
        end
        x2(i)=(b(i)-sum)/A(i,i);
    end
    r=max(abs(x2-x1));
    x1=x2;
    k=k+1;
    if k>N
        disp('迭代失败,返回');
        return;
    end
end
x=x1;
```

算例:用 Gauss-Seidel 迭代法求解方程组:

$$\begin{cases} -4x_1 + x_2 + x_3 + x_4 = 1 \\ x_1 - 4x_2 + x_3 + x_4 = 1 \\ x_1 + x_2 - 4x_3 + x_4 = 1 \\ x_1 + x_2 + x_3 - 4x_4 = 1 \end{cases}$$

其精确解为 x=[-1, 1,-1, 1]'.

解答:

令 A=[-4,1,1,1;1, -4,1,1;1,1,-4,1;1,1,1,-4];b=[1,1,1,1]';x0=[0,0,0,0]';在命令窗口中输入:

```
[x,k]=Gaussmethod(A,b,x0,100,10^-5);
```

运行结果为:

```
k=21
x=
   -9.999 896 479 636 309e-001
    9.999 910 053 552 269e-001
   -9.999 921 847 613 638e-001
   -9.999 932 095 200 554e-001
```

由此可知,在同种精度下,Gauss-Seidel 迭代法比 Jacobi 迭代法收敛速度快. 一般来说,Gauss-Seidel 迭代法比 Jacobi 迭代法收敛得快,但有时反而比 Jacobi 迭代法慢. 而且,Jacobi 迭代法易于并行化,因此,两种方法各有优缺点,使用时要根据需求选用.

文件 5.3　超松弛迭代

文件功能:用超松弛(SOR)迭代法求解线性方程组.

文件名:SORmethod.m

MATLAB 文件:

```
function [x,k] = SORmethod(A,b,x0,N,emg,w)
%A 是线性方程组的左端矩阵
%b 是右端向量
%x0 是迭代初始值
%N 表示迭代次数上限,若迭代次数大于 N,则迭代失败
%emg 表示控制精度
%w 表示松弛因子
%用 SOR 迭代法求线性方程组 A * x = b 的解
%k 表示迭代次数
%x 表示用迭代法求得的线性方程组的近似解
n = length(A);
x1 = zeros(n,1);   x2 = zeros(n,1);
x1 = x0;
r = max(abs(b - A * x1));
k = 0;
while r>emg
    for i = 1:n
        sum = 0;
        for j = 1:n
            if j> = i
                sum = sum + A(i,j) * x1(j);
            elseif j<i
                sum = sum + A(i,j) * x2(j);
            end
        end
        x2(i) = x1(i) + w * (b(i) - sum)/A(i,i);
    end
    r = max(abs(x2 - x1));
    x1 = x2;
    k = k + 1;
```

```
        if k>N
            disp('迭代失败,返回');
            return;
        end
    end
    x = x1;
```

算例:用超松弛(SOR)迭代法求解方程组:

$$\begin{cases} 4x_1 + x_2 + x_3 + x_4 = 1 \\ x_1 - 4x_2 + x_3 + x_4 = 1 \\ x_1 + x_2 - 4x_3 + x_4 = 1 \\ x_1 + x_2 + x_3 - 4x_4 = 1 \end{cases}$$

其精确解为 x-[-1,-1,-1,-1]'.

解答:

令 A-[-4,1,1,1;1,-4,1,1;1,1,-4,1;1,1,1,-4];b=[1,1,1,1]';x0=[0,0,0,0]';在命令窗口中输入:

```
[x,k] = SORmethod(A,b,x0,100,10^-5,1);
```

则运行结果为:

```
k = 21
x -
  -9.999 896 479 636 310e-001
  -9.999 910 053 552 269e-001
  -9.999 921 847 613 639e-001
  -9.999 932 095 200 554e-001
```

若输入:

```
[x,k] = SORmethod(A,b,x0,100,10^-5,1.25);
```

则运行结果为:

```
k - 10
x =
   1.000 002 971 098 328e+000
  -9.999 983 317 698 703e-001
  -1.000 000 777 664 050e+000
  -1.000 000 724 906 550e+000
```

由实验结果可知:当松弛因子为 1 时,超松弛迭代法等同于 Gauss-Seidel 迭代法,这和理论推导完全相同.另外,超松弛迭代法的收敛速度完全取决于松弛因子的选取,一个适当的松弛因子能大大提高收敛速度.

文件 5.4 对称超松弛迭代

文件功能:用对称超松弛(SSOR)迭代法求解线性方程组.

文件名:SSORmethod.m

MATLAB 文件:

```
function [x,k] = SSORmethod(A,b,x0,N,emg,w)
%A 是线性方程组的左端矩阵
%b 是右端向量
%x0 是迭代初始向量
%N 表示迭代次数上限,若迭代次数大于 N,则迭代失败
%emg 表示控制精度
%w 表示松弛因子
%用 SSOR 迭代法求线性方程组 A*x=b 的解
%k 表示迭代次数
%x 表示用迭代法求得的线性方程组的近似解
n = length(A);
x1 = zeros(n,1);   x2 = zeros(n,1);   x3 = zeros(n,1);
x1 = x0;   k = 0;
r = max(abs(b - A*x1));
while r > emg
    for i = 1:n
        sum = 0;
        for j = 1:n
            if j > i
                sum = sum + A(i,j)*x1(j);
            elseif j < i
                sum = sum + A(i,j)*x2(j);
            end
        end
        x2(i) = (1-w)*x1(i) + w*(b(i)-sum)/A(i,i);
    end
    for i = n:-1:1
        sum = 0;
        for j = 1:n
            if j > i
                sum = sum + A(i,j)*x3(j);
            elseif j < i
                sum = sum + A(i,j)*x2(j);
            end
        end
        x3(i) = (1-w)*x2(i) + w*(b(i)-sum)/A(i,i);
```

```
        end
        r = max(abs(x3 - x1));
        x1 = x3;
        k = k + 1;
        if k > N
            disp('迭代失败，返回');
            return;
        end
    end
    x = x1;
```

算例：使用对称超松弛迭代法和超松弛迭代法求解线性方程组：

$$\begin{cases} 4x_1 - x_2 = 1 \\ -x_1 + 4x_2 - x_3 = 4 \\ -x_2 + 4x_3 = -3 \end{cases}$$

精度控制在 10^{-5}.

解答：

令 A=[4, -1,0; -1,4, -1;0, -1,4];b=[1,4, -3]';x0=[0,0,0]';在命令窗口中输入：

```
[x1,k1]=SORmethod(A,b,x0,100,10^-5,1.2);
[x2,k2]=SSORmethod(A,b,x0,100,10^-5,1.2);
```

运行结果如下：

```
k1 = 9
x1 =
    4.999 979 586 011 558e-001
    9.999 998 363 532 925e-001
      4.999 999 992 142 435e-001
k2 = 7
x2 =
    5.000 001 207 146 788e-001
    9.999 993 572 021 568e-001
    -4.999 994 074 112 435e-001
```

对于用超松弛迭代法和对称超松弛迭代法求解方程组，关键在于松弛因子的选取，当松弛因子相同时，两者的收敛速度相当.但是，对称超松弛迭代法的预处理矩阵是对称阵，从这方面来说，它比超松弛迭代法的预处理矩阵好，并在预处理共轭梯度法中广为应用.

第六章　线性方程组的直接法

文件 6.1　追　赶　法

文件功能：用追赶法解三对角线性方程组 $\boldsymbol{Ax}=\boldsymbol{f}$.

文件名：threedia.m

MATLAB 文件：

```
function x = threedia(a,b,c,f)
%求解线性方程组 Ax=f,其中 A 是三对角阵
%a 是矩阵 A 的下对角线元素 a(1)=0
%b 是矩阵 A 的对角线元素
%c 是矩阵 A 的上对角线元素 c(N)=0
%f 是方程组的右端向量
N=length(f);
x=zeros(1,N);   y=zeros(1,N);
d=zeros(1,N);   u=zeros(1,N);
%预处理
d(1)=b(1);
for i=1:N-1
      u(i)=c(i)/d(i);
      d(i+1)=b(i+1)-a(i+1)*u(i);
end
%追的过程
y(1)=f(1)/d(1);
for i=2:N
      y(i)=(f(i)-a(i)*y(i-1))/d(i);
end
%赶的过程
x(N)=y(N);
for i=N-1:-1:1
      x(i)=y(i)-u(i)*x(i+1);
end
```

算例：用追赶法求解方程组：

$$\begin{pmatrix} 2 & -1 & & 0 \\ -1 & 3 & -2 & \\ & 1 & 2 & -1 \\ 0 & & -3 & 5 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}=\begin{pmatrix} 6 \\ 1 \\ 0 \\ 1 \end{pmatrix}$$

解答：

令 a=[0,-1,-1,-3];b=[2,3,2,5];c=[-1,-2,-1,0];f=[6,1,0,1]';在命令窗口运行语句：

x=threedia(a,b,c,f);

得结果为：

```
x =
   5   4   3   2
```

文件 6.2　Cholesky 方法

文件功能：用 Cholesky 分解法解对称方程组 $\boldsymbol{Ax}=\boldsymbol{b}$.

文件名：Chol_decompose.m

MATLAB 文件：

```
function x = Chol_decompose(A,b)
%用 Cholesky 分解求解线性方程组 Ax=b
%A 是对称矩阵
%L 是单位下三角阵
%D 是对角阵
%对矩阵 A 进行三角分解：A=LDL'
N = length(A);
L = zeros(N,N);   D = zeros(1,N);
for i = 1:N
     L(i,i) = 1;
end
D(1) = A(1,1);
for i = 2:N
     for j = 1:i-1
          if j == 1
               L(i,j) = A(i,j)/D(j);
          else
               sum1 = 0;
               for k = 1:j-1
                    sum1 = sum1 + L(i,k) * D(k) * L(j,k);
               end
               L(i,j) = (A(i,j)-sum1)/D(j);
          end
     end
     sum 2 = 0;
     for k = 1:i-1
```

```
            sum 2 = sum 2 + L(i,k)^2 * D(k);
        end
        D(i) = A(i,i) - sum2;
    end
    %分别求解线性方程组 Ly = b;L' x = y/D
    y = zeros(1,N);
    y(1) = b(1);
    for i = 2:N
        sumi = 0;
        for k = 1:i - 1
            sumi = sumi + L(i,k) * y(k);
        end
        y(i) = b(i) - sumi;
    end
    x = zeros(1,N);
    x(N) = y(N)/D(N);
    for i = N - 1: - 1:1
        sumi = 0;
        for k = i + 1:N
            sumi = sumi + L(k,i) * x(k);
        end
        x(i) = y(i)/D(i) - sumi;
    end
```

算例:用 Cholesky 方法求解方程组:

$$\begin{cases} 4x_1 - 2x_2 + 4x_3 = 8.7 \\ -2x_1 + 17x_2 + 10x_3 = 13.7 \\ 4x_1 + 10x_2 + 9x_3 = -0.7 \end{cases}$$

解答:

令 A = [4, -2,4; -2,17,10;4,10,9];b = [8.7,13.7, -0.7];在命令窗口中运行:

x = Chol _ decompose(A,b);

其运行结果为:

```
x =
    -5.145 7    -3.172 7    5.734 4
```

文件 6.3 矩阵分解方法

文件功能：基于 Gauss 消去法的 *LU* 分解求解线性方程组 $\boldsymbol{Ax}=\boldsymbol{b}$.

文件名：lu_decompose.m

MATLAB 文件：

```
function x = lu_decompose(A,b)
%基于矩阵的 LU 分解求解线性方程组 Ax = b
%A 表示系数矩阵
%b 表示方程组右边的向量
n = length(b);
L = eye(n);     U = zeros(n,n);
x = zeros(n,1);     y = zeros(n,1);
%对矩阵 A 进行 LU 分解
for i = 1:n
    U(1,i) = A(1,i);
    if i = = 1
        L(i,1) = 1;
    else
        L(i,1) = A(i,1)/U(1,1);
    end
end
for i = 2:n
    for j = i:n
        sum = 0;
        for k = 1:i - 1
            sum = sum + L(i,k) * U(k,j);
        end
        U(i,j) = A(i,j) - sum;
        if j ~ = n
            sum = 0;
            for k = 1:i - 1
                sum = sum + L(j + 1,k) * U(k,i);
            end
            L(j + 1,i) = (A(j + 1,i) - sum)/U(i,i);
        end
    end
end
%解方程组 Ly = b
```

```
y(1) = b(1);
for k = 2:n
    sum = 0;
    for j = 1:k - 1
        sum = sum + L(k,j) * y(j);
    end
    y(k) = b(k) - sum;
end
%解方程组 Ux = y
x(n) = y(n)/U(n,n);
for k = n - 1: - 1:1
    sum = 0;
    for j = k + 1:n
        sum = sum + U(k,j) * x(j);
    end
    x(k) = (y(k) - sum)/U(k,k);
end
```

算例:求解方程组:

$$\begin{pmatrix} 0.001 & 2 & 3 \\ -1 & 3.712 & 4.623 \\ -2 & 1.072 & 5.643 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

解答:

令 A = [0.001,2,3; - 1,3.712,4.623; - 2,1.072,5.643];b = [1,2,3];在命令窗口中运行:

```
x = lu_decompose(A,b);
```

其结果为:

```
x =
    -0.490 4
    -0.051 0
    0.367 5
```

文件 6.4　消　去　法

文件功能:用 Gauss 列主元消去法求解线性方程组 $Ax = b$.

文件名:Gauss_pivot.m

MATLAB 文件:

```
function x = Gauss_pivot(A,b)
%用 Gauss 列主元消去法解线性方程组 Ax = b
%x 是未知向量
```

```
n = length(b);
x = zeros(n,1);
c = zeros(1,n);
d1 = 0;
for i = 1:n-1
    max = abs(A(i,i));
    m = i;
    for j = i+1:n
        if max < abs(A(j,i))
            max = abs(A(j,i));
            m = j;
        end
    end
    if(m ~= i)
        for k = i:n
            c(k) = A(i,k);
            A(i,k) = A(m,k);
            A(m,k) = c(k);
        end
        d1 = b(i);
        b(i) = b(m);
        b(m) = d1;
    end
    for k = i+1:n
        for j = i+1:n
            A(k,j) = A(k,j) - A(i,j) * A(k,i)/A(i,i);
        end
        b(k) = b(k) - b(i) * A(k,i)/A(i,i);
        A(k,i) = 0;
    end
end
%回代求解
x(n) = b(n)/A(n,n);
for i = n-1: -1:1
    sum = 0;
    for j = i+1:n
        sum = sum + A(i,j) * x(j);
    end
```

```
        x(i) = (b(i) - sum)/A(i,i);
    end
```

算例:求解方程组 $\boldsymbol{Ax}=\boldsymbol{b}$,其中

$$\boldsymbol{A}=\begin{bmatrix}0.3\times10^{-15} & 59.14 & 3 & 1\\ 5.291 & -6.13 & 1 & 2\\ 11.2 & 9 & 5 & 2\\ 1 & 2 & 1 & 1\end{bmatrix},\quad \boldsymbol{b}=\begin{bmatrix}59.17\\ 46.78\\ 1\\ 2\end{bmatrix}$$

分别用 Gauss 列主元消去法和 Gauss 消去法求解.

解答:

令 A=[0.3 * 10^ - 15,59.14,3,1;5.291, - 6.13, - 1,2;11.2,9,5,2;1,2,1,1];b=[59.17,46.78,1,2]';在命令窗口中运行:

```
x = Gauss_pivot(A,b);
```

其结果为:

```
x =
   3.845 714 853 511 634e+000
   1.609 517 394 778 522e+000
     1.547 605 454 206 655e+001
   1.041 130 489 899 787e+001
```

此时

```
b - A * x =
                                0
                                0
         3.552 713 678 800 501e-015
         1.776 356 839 400 251e-015
```

由此可知,用 Gauss 列主元消去法所求得的解相当准确.

在命令窗口中运行:

```
x = lu_decompose(A,b);
```

得

```
x =
                          0
     5.101 454 176 530 265e-001
     2.500 000 000 000 002e+001
     4.600 000 000 000 004e+001
```

此时

```
b - A * x =
                                  0
           1.669 071 914 102 132e+002
          -3.659 130 875 887 726e+001
           2.197 970 916 469 397e+001
```

由此可知,对于此方程组,直接用 Gauss 消去法求解不能得到准确解,其原因是主元素的值非常的小,因此,对于一般的方程组,应尽量使用带有主元的 Gauss 消去法求解.

结　束　语

随着科学技术的进步,MATLAB 在科学计算中的应用范围越来越广.由于它简单易学且易于做数值试验,已受到广大数学爱好者的青睐.认为 MATLAB 将取代其他高级语言(如 C 或 FORTRAN)或许还言之过早,但在某些方面,说它无可比拟则一点也不言过其实.在数值试验和模拟中,若用一般的高级语言(如 C 或 FORTRAN),人们常常为一些小细节而绞尽脑汁,而用 MATLAB,一两行语句往往就能轻易地解决问题.

由于市面上已有大量的 MATLAB 书籍出现,且本附录中的大部分文件都只需要运用 MATLAB 的基本操作,因此关于 MATLAB 简介部分不在附录中给出,读者可参考任何关于 MATLAB 操作的书籍.只需掌握 MATLAB 的一些基本操作,就会编写一些基本的 MATLAB 程序.这正是 MATLAB 的诱人之处.当然,优化程序设计需要更多的 MATLAB 操作和程序实践知识.凡事都有一个循序渐进的过程,从最简单的做起,并脚踏实地坚持下去,就一定能逐步提高自己的编程能力.